DEVELOPMENT AND EVALUATION OF A SECURE WEB GATEWAY WITH MESSAGING FUNCTIONALITY

Utilizing Existing ICAP and Open-source Tools To Notify And Protect End Users from Internet Security Threats.

A thesis submitted in partial fulfillment of the requirements
for the Degree of Master of Science
in
Computer Science at the University of Canterbury
by

Michael Bruce Pearce

University of Canterbury
Department of Computer Science and Software Engineering
2010

Abstract

Secure web gateways aim to protect end user systems against web based threats. Many proprietary commercial systems exist. However, their mechanisms of operation are not generally publicly known. This project undertook development and evaluation of an open source and standards based secure web gateway. The proof of concept system developed uses a combination of open source software (including the Greasyspoon ICAP Server, Squid HTTP proxy, and Clam Antivirus) and Java modules installed on the ICAP server to perform various security tasks that range from simple (such as passive content insertion) to more advanced (such as active content alteration).

The makeup of the proof of concept system and the evaluation methodology for both effectiveness and performance are discussed. The effectiveness was tested using comparative analysis of groups of self-browsing high interaction client honey pots (employing a variety of security measures) and recording different system alteration rates. Performance was tested across a wide range of variables to determine the failure conditions and optimal set up for the components used.

The system developed met the majority of the goals set, and results from testing indicate that there was an improvement in infection rates over unprotected systems. Performance levels attained were suitable for small scale deployments, but optimization is necessary for larger scale deployments.

Acknowledgments

Many people were bombarded with various questions, and had ideas posed to them during the course of this work. While I cannot mention everyone by name, I further appreciate the presence and support of the many other students and staff around the department who were were working around, near, and with me.

My supervisor Ray Hunt was consistently either a knock or an email away, for which i am very grateful. Malcolm Shore was always ready to cast a skeptical eye across my ideas and work, and to suggest somewhere else i could consider taking it.

I must also give special thanks to the others working in my lab, as they were the first to receive my rants when things went awry. When all else failed they could mock me for the number of computers surrounding me.

I could not have completed this endeavour without the generous and (sometimes) gracious assistance of my wife Lucy, who was exposed to a barrage of grammatical questions, queries, and scenarios over the last few months. She generally had an answer, HOWEVER some of them she even had to look up! ;)

Michael Pearce
December 2010

Peer Reviewed Papers on this Work

Michael Pearce, Ray Hunt, *Development and Evaluation of a Secure Web Gateway Using Existing Open Source and ICAP Tools.*, Presented at 10th SECAU Security Congress, Perth, Australia, 30 Nov - 2 Dec 2010.

Contents

Glossary **1**

1 Introduction **3**
1.1 Project Overview . . . 3
1.2 Motivation . . . 3
1.3 Background . . . 4

2 Technical Introduction **17**
2.1 HTTP . . . 17
2.2 Web Content . . . 19
2.3 ICAP . . . 27
2.4 Related Technologies . . . 36
2.5 Relevant Threats . . . 40
2.6 Protecting Clients . . . 44
2.7 Related Research . . . 51

3 System Development **59**
3.1 Goals . . . 59
3.2 Hypotheses and Assumptions . . . 63
3.3 Novel Aspects . . . 64
3.4 Requirements . . . 65
3.5 System Architecture . . . 67
3.6 Main Component Details . . . 69
3.7 Component Configurations . . . 72
3.8 General Script Operations . . . 74
3.9 Specific Operations Implemented . . . 75

4 Testing Methodology **89**
4.1 Testing Overview . . . 89
4.2 Effectiveness Testing . . . 90
4.3 Performance testing . . . 99

5 Results **103**
5.1 Functionality of System Developed . . . 103
5.2 Requirement Evaluation . . . 107
5.3 Effectiveness Testing . . . 112
5.4 Performance Testing . . . 125

6 Conclusions **135**
6.1 Project Summary . . . 135

7 Future Work **139**
7.1 Developed System - Fixes and Optimization . . . 139
7.2 Developed System - Enhancements and extensions . . . 139
7.3 Effectiveness Testing . . . 142
7.4 Performance Testing . . . 142

List of Figures **144**

List of Tables **145**

References and Bibliography **146**

A The HyperText Transfer Protocol **160**
A.1 HTTP . . . 160

B Testbed Component Layout **168**

C Comcast System Requirements in full **170**

D Threat Surveys and Studies **172**

E Configuration Settings **173**
E.1 System Settings . . . 173

F Testing Configuration **176**
F.1 Effectiveness Testing . . . 176
F.2 Performance Testing Automation . . . 179

G Database Schemas **181**
G.1 usermessages . . . 181

H Example Greasyspoon Code **182**
H.1 Helper Code . . . 182
H.2 Sample GreasySpoon Scripts . . . 183

I Sample Results **189**
I.1 Sample Drive by download steps taken from an Capture-HPC Log in the experiment . . . 189
I.2 ApacheBench Data . . . 191

Glossary

APWG Anti-Phishing Working Group.

ASLR Address Space Layout Randomization.

Caching Storing something for ease of access later.

CSS Cascading Style Sheets.

DDOS Distributed Denial of Service.

DEP Data Execution Protection.

DOM Document Object Model.

Fast Flux Network a network that is hosted upon or proxied through a fast changing arrangement of compromised hosts..

Fingerprinting assessing details about the target.

FTP File Transfer Protocol.

GPL GNU General Public License.

HTML Hyper Text Markup Language.

HTTP Hyper Text Transfer Protocol.

HTTPS Hyper Text Transfer Protocol (Secure).

ICAP Internet Content Adaptation Protocol.

IETF Internet Engineering Task Force.

IP Internet Protocol.

ISP Internet Service Provider.

MAC Mandatory Access Control.

NAT Network Address Translation.

PCIDSS Payment Card Industry Data Security Standard.

RFC IETF Request For Comment.

SGML Standard Generalized Markup Language.

TCP Transmission Control Protocol.

TLD Internet Service Provider.

UDP User Datagram Protocol.

URL Uniform Reference Locator.

W3C World Wide Web Consortium.

WMF Windows MetaFile.

WWW World Wide Web.

Zero Day Vulnerability A vulnerability for which no patch is yet available..

Chapter 1

Introduction - Motivation and Background

1.1 Project Overview

This project undertook to build a secure web gateway entirely out of open source software. Many of the concepts in a technical note of a web user notification system [22] (which has since been deployed into production by the American Internet Service Provider Comcast) were incorporated into the design, and the associated ideas extended upon to form a system that performed security mitigation measures. The system's effectiveness against malicious downloads was evaluated, followed by testing of the performance impacts (measured by content throughput and delay) of the various security operations the system performed.

1.2 Motivation

World Wide Web (WWW) (or web) [1]-based technology and tools are increasingly one of the key media and information sources around the world. Furthermore, there has been a trend in recent years of decreasing use of dedicated desktop applications in favor of increased use of web-based applications and browser-based tools for both home and enterprise use. The secure undertaking of affairs online is dependent upon many different factors, but the web facing application – the web browser – is the common link across all web applications. Web browsers are both exposed

[1] "Web" content is carried on high layers of the Internet, and care will be taken in this work to use the terms "Web", "WWW", "HyperText Transfer Protocol (HTTP)", or "https" when referring to using those protocols (ISO layer 5 and up[ref]) and "Internet" when referring to the Internet as a whole or its underlying communication and transport protocols such as Internet Protocol (IP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) (ISO layers 1-4)

to the greatest number of users and the most used class of software on computers[12], making them especially important.

The web is also increasing in interactive and dynamic content through "Web 2.0" and similar technologies and approaches. The increased use of technologies such as JavaScript, AJAX, Flash, and Silverlight has resulted in expansion of the functionality websites offer. Where early websites were initially only sources of static data and content, content that is dynamic and executable is now often provided and used. As a result, instead of being simple HTML renderers, web browsers are now their own software environment[5]. Whilst enabling many new and exciting behaviors and solutions to problems, this also greatly increases the potential attack surface.

Based upon the HTTP and HyperText Transfer Protocol (Secure) (HTTPS) protocols, web content has become nearly ubiquitous in the modern world. Its success has come about because the underlying protocols, HTTP and HTTPS are flexible, simple, human readable protocols that, whilst designed for primarily textual content, can be used to carry any type of data. We shall discuss these technologies in more detail in the Technical Introduction (chapter 2).

The content-agnostic approach of the web has resulted in an explosion not only in the numbers of its users, but also in the number of its uses. Unfortunately, these same principles of flexibility and openness that make the web so flexible can also leave web users (and their computer systems) open to abuse.

The security of web activities has become important for day to day affairs, critical business, and national security. Almost every class of organization and government agency uses the web to transport information of varying sensitivity. The compromise, loss, or corruption of that information can result in losses that range from trivial to potentially life-threatening.

Undesirable Internet content that uses HTTP as an attack vector is increasingly affecting end users and can be broken down into two main forms which we discuss next:

- Objectionable human targeted content, such as fraudulent or illegal content
- Dangerous computer targeted content, such as viruses, worms, trojans and spyware

1.3 Background

1.3.1 Problematic human targeted content

Problematic content targeted at people (not the computer) comes in various forms. The relevant forms of such content to the web domain are those inappropriate due to the context, and those

that facilitate fraud against the user.

Inappropriate Content

Whether content is classed as inappropriate is highly dependent upon the relevant context. Appropriateness can be very contentious, as it often comes down to religious, political, legal, jurisdictional, or other subjective criteria. Common examples of content that are often considered inappropriate in western organizations include hate speech and pornographic content, while in other parts of the world politics, religion, or any form of sexual content can be considered inappropriate in every context. Less controversial examples of inappropriate content are games or social networking sites in the workplace, which employers may consider a hindrance to productivity. This work, however, is concerned with the more general technical issues. As dealing with inappropriate content is peripheral to information security, it will be addressed only briefly.

Fraud Driven Content

The other class of problematic content targeted at human users, content which facilitates fraud against users, is far more clear cut and is a real risk to the computer or information security of users. We shall use the term ”social engineering” because of the way these attacks are normally undertaken: through the use of some normal behavior, assumption, or expectation to manipulate the user. Two classic attacks fall into this category: disreputable or fraudulent retailers and phishing or site impersonation.

Disreputable or Fraudulent Retailers and Websites Much as in the physical world, websites have varying levels of reputability. Some can presumably be trusted with one's entire net worth (such as your bank), whilst you are better off with others not even knowing your name. Also, like the physical world, disreputable types are often located together – in certain Top Level Domain (TLD)s such as .cc, .cm, or .cn[104][83]

Disreputable sites can be dangerous to users in various ways, but common examples include fake storefronts that don't actually deliver anything, the delivery of illegal or contaminated goods (such as banned pornography, or pharmaceuticals, pirated software, or malware-infected software), or the harvesting of user information. This can result in identity theft, legal liability, illegal activity, or even death in some cases of fake pharmaceuticals bought online [159].

While the real world metaphor holds in some ways, it is deceptive in others. For instance, while there are higher proportions of disreputable sites in the dangerous TLDs, there are more

by number in the very populated ones such as .com, with the result that it can be very difficult for users to detect dangerous groups of sites by address alone.

Unfortunately, other indicators often do not offer much help either. For instance, people often judge the trustworthiness of a retailer by the amount of perceived quality and effort that has gone into their storefront[54], but on the web (as often in real life) this is not a reliable indicator. It can even be counter-intuitive, because reputable retailers have to deal with more regulations and costs such as Payment Card Industry Data Security Standard (PCIDSS) and Health insurance Portability and Accountability Act (HIPAA) requirements, whereas scam sites need only worry about getting the information they need as quickly as possible from the user. Among the most valuable pieces of information are financial credentials such as credit card information, but the target can often be something as simple as an email address, which can then be sold. Irrespective of the sensitivity of this information, it is undesirable for it to become compromised and inconvenience the owner.

So, due to the difficulty in assessing the trustworthiness of web content from the site content, a common approach is to look for user reviews from others who have dealt with the site. However, it can be very difficult to assess the validity of the reviews themselves, with numerous reports of retailers posting their own fake reviews on review sites. But these measures often will not help at all if the user thinks it is a legitimate site they would normally visit such as their bank or email.

Impersonation of legitimate websites is very simple to do online, because the markup needed to display a page is, by necessity, sent to users. An attacker can copy the appearance of a site simply by taking this code and putting it on a web server controlled by the attacker. By changing the target address the form data is sent to, anything input by a user can be captured, and if the attack is sophisticated it can then be submitted to the real site so the victim never knew anything was awry.

Users are encouraged to look for "the "padlock" icon or HTTPS URL prefix when they are on a website, but this itself is not guarantee of anything regarding the identity of the site, as many legitimate websites use bad security practices. An example of such a bad practice is using self-signed certificates[137]. This offers encryption, but no assertion of identity whatsoever, as a self-signed certificate can be easily signed by an attacker[66]. Furthermore, browsers offer weak differentiation between certificate types, so users often accept them anyway[137]. The address bar in Internet Explorer's latest version is shown below: note that it distinguishes sites using Extended Validation certificates from those using other types of valid certificates by giving

the extended validation address a green background. However, both sites have a padlock, so an inexperienced user will have difficulty in knowing what the difference is.

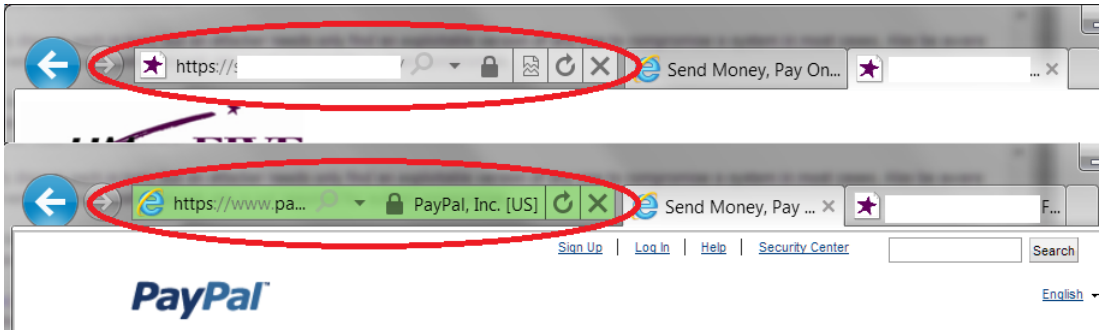

Figure 1.1: The address bar in Internet Explorer's latest version (9 Beta), with the difference between sites using Extended Validation certificates and those using other types of valid certificates circled.

The difficult part is to get the user to visit the fake site in the first place. The methods used to undertake this are usually collectively referred to as "phishing".

Phishing is described by Microsoft as "A method of identity theft that tricks Internet users into revealing personal or financial information online[103], while the Anti-Phishing Working Group (APWG) has a broader definition: "a criminal mechanism employing both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials The APWG definition includes what we term 'hybrid attacks', which we discuss later. Phishing emails are a high priority in the blocking and filtering undertaken by email providers (along with spam.) Commonly impersonated sites are financial institutions, social networks, and e-commerce sites[101]. The APWG reports that more than half of all targets were financial for the 3rd quarter of 2009, because they are the easiest to monetize[6] and this pattern is also reported more generally by other groups.

Some common methods to get users to visit phishing websites are misleading emails, misleading links, and misspelled urls as well as more advanced techniques such as cross-site scripting. Misleading urls have the text set to one link, but can (usually) be seen to actually link to another address when hovered over. Various methods are used to fool users into clicking these links; one common approach involves using a confusingly similar looking domain, for instance, by using http://www.paypaI.com[57]. However it gets more complicated when non-Latin characters are involved, in so called homograph attacks, as then the Uniform Reference Locator

(URL) is visually indistinguishable from the legitimate one[71].

The hosts for phishing websites can be anywhere, but tend to follow the pattern for disreputable sites in general and be more concentrated in smaller and riskier locations, but more numerous in absolute terms in the big TLD blocks (eg.com)[101]. The APWG reports that hosts in the USA consistently contain the most phishing sites.[6]

1.3.2 Dangerous computer targeted content

Due to the way web browsers are designed to work (in an untrusted environment running untrusted code and unchecked data), they can be exposed to code and data from a very large number of sources. For example, if a user goes to only 20 web sites in a day, each containing code for advertising, widgets and included media, then the user could be exposed to code from hundreds to thousands of subdomains.

Consider this: at the time of writing (September 2010), the main page on cnn.com[32] had around 60 JavaScript elements (including inline) from 12 different subdomains, of which only 3 appeared to be CNN-controlled. The others included social networking sites, advertising sites, survey delivery sites, and content delivery partner sites. If only one of those sources is compromised, then the user is at risk. Recent examples of this occurring include websites such as The New York Times[116] (see figure 1.3.2), Fox News[38], Whitepages[96], Gawker[127], and Trademe[52]. While prevalence of infection in legitimate sites is seemingly low, at fractions of one percent for popular websites (as discussed in section 2.7.1), the sheer number of code sources makes exposure to malicious code far more likely.

There are two main risks from compromise by untrusted code: it can mislead users, or it can exploit their software. Both are damaging. Malicious code from the web targets one of four attack surfaces:

- The Operating System
- The Web Browser
- Web Browser Plugins
- Other Web Aware Software

An attacker needs only find an exploitable version of any one to compromise a system in most cases. User intervention beyond visiting a web site is not required for system compromise.

Note: Discussion of software measures for mitigating the exploitation of software is included as background only. This thesis is concerned chiefly with methods of preventing

Figure 1.2: Personal Antivirus -a fake antivirus that was delivered through malicious ads on the New York Times webpage in 2009[176]

dangerous code making its way on to the system where the vulnerability lies.

Attacks Targeting the Operating System

Attacks targeting the operating system are generally the most dangerous, but are becoming rarer as operating systems are becoming more secure. Attacks of this type take advantage of an operating system weakness, usually in some shared library, and so can affect systems of many different configurations. While all operating systems have been targeted by such attacks, we shall limit our discussion here to Microsoft Windows, as it is the most commonly used and attacked operating system at present. Notable examples of Windows operating system vulnerabilities that have been used to attack web users are the GDI+ (MS04-028 vulnerability [99]), Windows MetaFile (WMF)(MS06-001 vulnerability [100]), and more recently the .lnk (link) file format vulnerabilities (CVE-2010-2568 [36]). These vulnerabilities were exploited by attackers simply by attempting to display a maliciously crafted file on a website in any software that used the appropriate Windows libraries. These all affected Internet Explorer, and other web browsers to various degrees. The problems were exacerbated in early versions of Internet Explorer, which had fewer separation techniques applied to isolate it from the Windows operating system than more recent versions.

Attacks Targeting the Web Browser

As the operating system libraries have become less easy to attack, the web browsers became the next step. Although they add another layer of variation upon the operating system there are still only four to five main players on the desktop, and they generally disclose their version with every page request which makes correct identification trivial. Furthermore, because most users log in and run software as an administrative user, compromising the web browser process allowed full system compromise. The addition of protection technologies such as sandboxing[2], Stack Cookies[3], Data Execution Protection (DEP) and Address Space Layout Randomization (ASLR) have made the exploitation of browsers much more difficult, but not all browsers yet implement such measures[5]. Furthermore, even when such protections are implemented, there are measures of evading them and still exploiting vulnerabilities that exist in the browser[160].

The best way to protect browsers from exploitation is to ensure there are no exploitable bugs in the first place. All of the memory protection technologies are designed to make exploitation of bugs difficult, but they do not prevent the existence of the bug in the first place. The term *Zero Day* is used to describe exploits of bugs for which there is no patch available. Exploitable zero day bugs can be sold for thousands of dollars. A concerted effort has been made by browser vendors to harden their products with each release, and to roll out patches to problems promptly. Vendor awareness of the importance of repairing such problems before they become public has also lead to some vendors offering so-called *bug bounties* of thousands of dollars for disclosure and confidentiality of bugs found by others in their products [112] [47].

Attacks Targeting Browser Plugins

As web browsers have become more difficult to successfully exploit, attackers have changed target yet again to more easily exploited classes of software: plugins, extensions, and addons that run inside (or alongside) web browsers. Even though the browser may be sandboxed, by linking to or running third party software in browser plugins, it is possible to infect an operating system via weaknesses in that software. This linkage is illustrated in figure 1.3.

This additional layer of variation upon the systems again offers only marginal security, as plugin versions installed upon visitor's systems are easily determined by web sites. Furthermore, much like web browsers, there are several plugins that most users can be reasonably expected

[2] Sandboxing refers to the execution of software in an isolated and controlled environment where it cannot affect anything outside of the isolated environment (*sandbox*) without permission

[3] A *stack cookie*, or /gs protection adds a marker at the end of data portions of code that should never be modified, and raises an alert if it is modified

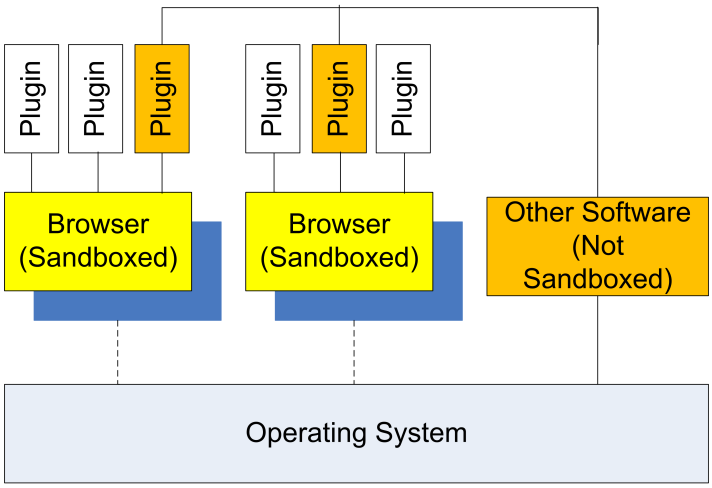

Figure 1.3: A simplified software hierarchy illustrating the linkages that may exist between web browser plugins and third party software

to have installed. Those which about half or more users have installed include some variant of: Flash, Java, Windows Media Player, Quicktime, Shockwave, Realplayer, Silverlight, and Adobe PDF[56][165]. Of those the two particularly dangerous ones are also the most prevalent[5]: Adobe's Flash and PDF, which together accounted for over 95% of exploits in 2009[145].

Attacks Targeting Other Web Aware Software

Any software that uses the web can be theoretically exploited, but that which is of those most risk to web browser users (other than that using plugins) is usually that which uses url-handlers. A url handler is what the operating system calls for an associated protocol in a URL/URL. For example, http://... will be handled by a web browser, whereas if iTunes is installed then iTunes will be invoked for for itms://... URLs. Using these it is possible to open programs and open files with those programs with no user intervention[4]. Combined with buffer overflows this has been used for exploiting systems in the past [35].

1.3.3 Combined Attacks (Utilize both Exploit and Social Engineering)

While both social and technical attacks are dangerous in their own right, they are a far greater threat when used in combination. This form of attack is becoming more prevalent as users become more aware of (and thus less likely to be vulnerable to) simple types of attacks such as those purely technical (ie trojan key-loggers) or those fully social (ie phishing sites)[190]. Although there is no limit to the combinations that can be used for these attacks, common com-

[4] Although later versions of web browsers warn before automatically calling external URI handlers

bined social and technical attacks use one method to infect and another to gain the information what they want. So, they use social measures to spread/infect and technical to accomplish their aim, or technical to infect and social to accomplish their aim.

Combined social/technical attacks are the more commonly encountered, these are the attacks which try to convince the user to bypass technical restrictions to install malware. This can range from simply downloading and installing the software (thus bypassing the web browser sandbox), to disabling anti-virus, firewall, web browsers and operating system security warnings before downloading and installing. Once on the system the technical aspects of the attack take over, and the software acts as a rootkit, keylogger, backdoor, or otherwise.

Technical then social attacks are more difficult to stop, and are reportedly one of the more profitable forms of attack at present[ref]. The most popular form of this is fake anti-virus[ref], which once installed uses software of a similar look, feel, and behavior to legitimate anti-virus products in an attempt to convince the customer to provide payment details to "upgrade to the full version", "subscribe for protection", "purchase the full version", or similar. Legitimate software produces very similar warnings, some of which even have been accused of offering the same level of threat[62] The difference with fake anti-virus products is that they claim to remove threats detected after scanning the system. Not only do the threats not exist, the software offers no protection at all against threats and may come with software to installer further malware.

1.3.4 Technological and Usage Trends

There are also some other current general trends which also influence this project greatly. They affect the security landscape for the web, thus some of these are worth noting as they affect the threats, the vulnerability points, and how countermeasures can be deployed. These include: the blurring of the line between producer and consumer everywhere online, the popularity of mobile devices, and the rising sophistication of bot-nets.

Web 2.0 - and the blurring between content consumers and producers

Websites from trusted organizations are often bound by their brand (and the law) to provide content that is truthful and safe, however, many such websites have some form of comments feature enabled, and almost all have advertising. While not in themselves threats, any flaws in the implementation of these features can allow code from a third party to be inserted into the page, with the effect that the content of page itself cannot be trusted.

Social media sites can abuse user trust in a different way: by subverting the postings of

users it is possible for them to impersonate them online and exploit the trust afforded to them by other users. This can be used to get them to purchase goods, disclose personal information, download software, or many other actions they would not do had it not been recommended by their "friend".

Shortened URLs

Although URL shortening redirections services (see section 2.4.3) have been in existence for around ten years, the rising use of social media that is sensitive to message length (for example Twitter with its 140 character message size) has driven significant growth in the usage of such services. The prevalence of such links adds risks to users in several ways, but among the most important of these are that such links break the 1 to 1 property of URLs (where one URL points to one resource) and as such a user cannot know where a link is going before they click it. For an example see the image from Twitter below:

Figure 1.4: An example of a shortened link on a social networking site

There is no way for the average user to know what that link is before clicking it. This problem is exacerbated on mobile devices where typing or even displaying a full URL can be laborious at best due to the difficulty of text input and small screen size.

Popularity of mobile devices

The rising popularity of mobile devices increases the risks beyond those introduced by short URLs. Mobile devices such as netbooks, smartphones, and tablets have properties and are used in ways that make them in some ways more vulnerable than traditional desktop or notebook computers. Of particular note for this are the following issues:

- Limited system resource power (CPU, RAM, Storage)
- Battery life considerations
- Limited network capacity
- Only limited user access allowed

These each have effects upon the security environment of mobile devices.

System resources on mobile devices are often limited. This has security implications, as the effect of this is to make it undesirable for performance reasons to install client side protection measures and leave them running the in background. This is improving with more recent devices that have more power to spare, however the other issues given still act as constraints.

Battery life is another important consideration for mobile devices[133], and in the case of smartphones often one of the more important – after all, if a mobile phone goes flat you lose a major form of communication, an unacceptable and potentially dangerous situation for some users. Because every CPU clock cycle uses energy from a finite storage battery (as opposed to an effectively infinite electricity grid connection) mobile devices use low power modes when not performing operations. Every operation above the minimal consumes some energy from the battery that would not otherwise be consumed. This too is slowly improving with improving battery technology, but is still important.

Limited network capacity is another important consideration for mobile devices. As mobile devices often use the cellular network (when not on wifi or bluetooth) the bandwidth can be expensive and of a comparatively high latency. While desktop security often requires numerous software and signature updates, or lots of two way communication for cloud based approaches, these add unacceptable costs to mobile devices. This affects not only client based security solutions (such as antivirus or firewall), it also affects process such as regular automated patching. To a user with a 200MB per month data cap even 10MB can be very expensive if it pushes them over their data cap. Network capacity is improving, slowly improving this aspect of security also, but in many cases users are not able to install such patches or software anyway due to access limitations.

Limited user access is imposed upon many modern mobile devices, and particularly to those on cellular networks. Unlike on a home PC, users ordinarily cannot perform privileged operations at all – even to administer a device they own. Common smartphone operating systems including Google's Android and Apple's iPhone OS (iOS) come with root access disabled in most cases (and all cases on iOS). This offers some security, as programs are run Sandboxed (See section 2.6.1) and cannot access unauthorized system or program data, but it also disables users from installing security software, changing locked security settings, or applying patches that are not signed by the the vendor or network operator. It is possible to overcome this limitation by performing a process known as *jailbreaking* on iOS devices or *rooting* on Android

devices and attain full access to the device, but this is not only technically complicated or risky, but in some cases breaks contractual or legal obligations. There have been instances of rooted devices being both less secure (for example the iKee worm spreading on jailbroken iOS 3 devices (via SSH being default enabled with a default password[125]), and more secure (for example Android devices after installing Security Enhanced Linux, SELinux [153])

Botnets

Rajab et. al.[1] describe a Botnet as follows :

... [N]etworks of infected end-hosts, called bots, that are under the control of a human operator commonly known as a botmaster. ... The primary purpose of these channels is to disseminate the botmasters commands to their bot armies...[1]

Because there is money to be made in utilizing infected machines, machines compromised by many modern botnets can be intentionally very difficult to detect. They utilize rootkit and process hiding behavior, disable client side protections, and often cause no discernible difference in system performance. However, the entire system is open to the attackers should they wish to install malicious software, or use the system in an attack. Furthermore, often it can be difficult for an end user to prove they did not perform some action that occurred through their compromised system. This activity can include Distributed Denial of Service (DDOS), spam, website hosting, and even activity occurring through an installed web proxy.

Not only is a compromised machine a security risk for an end user, it is also a security risk for the network operator, be they corporate or an Internet Service Provider (ISP). For an ISP risks include increased network flow causing quality of service issues and bandwidth costs, potentially having their user's emails blocked, and even potentially being blacklisted from routing tables.

Chapter 2

Technical Introduction

The modern World Wide Web (WWW) is built upon many different technologies, but the most foundational of those are the markup language, HyperText Markup Language (HTML), and the transfer protocol, HyperText Transfer Protocol (HTTP). We will discuss these technologies as well as briefly addressing some of the additional technologies that are mentioned by or relevant to understanding our work. Note that we will only discuss ISO layer 5 and above technologies, as the underlying transport protocols are not generally relevant to the topic of this work. Proxied HTTP in particular will be discussed in detail, as the interworking of proxied HTTP and Internet Content Adaptation Protocol (ICAP) is crucial to this work.

2.1 HyperText Transfer Protocol (HTTP)

HTTP is the primary technology used by the WWW to transport information around. Defined and driven by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), it is a human readable text-based protocol initially designed for the retrieval of static text-based documents from remote servers, that has since been expanded to allow additional flexibility with regards to content types and better support for additional transfer operations such as proxying and caching. HTTP is defined in a series of IETF Request For Comment (RFC)s, but the one pertaining to the version in common use is defined in rfc 2616[51]. Knowledge of HTTP operation is assumed, but an overview of HTTP and its operation as relevant to our work is given in appendix A.1.1.

2.1.1 Proxied HTTP

Proxied HTTP is very similar to normal HTTP, but requests and responses are served via an intermediary server that understands and uses the HTTP protocol. This intermediary server is called an *HTTP proxy server*, a *Web proxy server*, or sometimes simply a *Web proxy* or even simply a *proxy* –with the type implied by the context.

A client requests that a web proxy perform an HTTP request (such as a GET request) to a server on behalf of the client, and forward the result to the client. Note that the HTTP specification[51] requires that clients using an HTTP proxy put the full resource URI in their requests, rather than simply relevant to the server root. For example:

GET www.google.com/ HTTP/1.1

instead of the non-proxied request line:

GET / HTTP/1.1

Proxied HTTP Operation

The operation of proxied HTTP operates in an architecture like that in figure 2.1, and consists of the following steps:

1. Client requests the content from the proxy.
2. Proxy requests the content from the web server itself
3. Proxy forwards desired content to the client

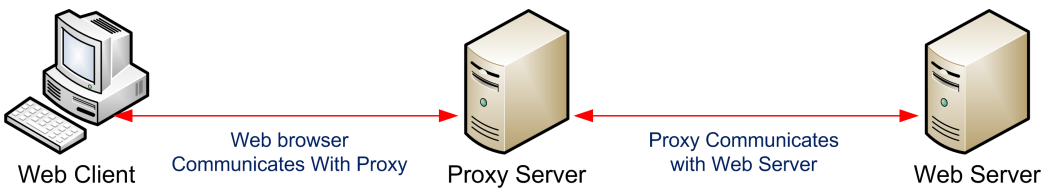

Figure 2.1: A proxied HTTP connection with a single proxy

Note that there can be multiple cascading levels of proxies, with a proxy requesting content via another proxy (and so on) – but the logic is the same as with forwarded requests.

Proxied HTTP also has the benefit of facilitating the Caching of responses, thus saving on resources and network bandwidth. However, we have disabled caching for this project; the effects of caching upon the system are left to future research (see section 7.1).

Transparent Proxied HTTP

Transparent Proxied HTTP looks to the client like an unproxied HTTP connection, but in reality the request is served via an inline proxy that transparently intercepts requests. This enables the use of a proxy without client configuration, and does not need the client to send the full absolute path with each request, as the proxy performs the necessary transformation.

2.1.2 Direct HTTPS

HTTPS, or HTTP with SSL uses asymmetric encryption and digital certificates to set up a secure and encrypted connection between an HTTPS web server and an HTTPS client. HTTPS is not used in this project because it is by design not easily possible to inspect the encrypted content without setting off errors, alerts, and warnings in web clients. There are ways around this, and many related non-technical issues also – but these are relegated to future work (see section 7.2.1).

2.1.3 Proxied HTTPS

When an HTTP proxy that is unable to decrypt and re-encrypt HTTPS receives an HTTPS request it should switch to tunnel mode, and simply pass on the raw content unmodified. This is usually undertaken via the CONNECT method. See section 7.2.1 for proposed details on secure connection termination and reencryption.

2.2 Web Content and Applications

The web has many associated technologies that use it, and many different types of static and active content are carried through HTTP. We shall briefly discuss a few of the most important types that relate to this project, and some of their associated security risks.

2.2.1 HTML

HyperText Markup Language, or HTML, defines the majority of textual web content and is the primary markup which the web was (and generally still is) built upon. HTML was originally proposed by Tim Berners-Lee in a document entitled *Information Management: A Proposal*[14] written to the management at the European Organization for Nuclear Research (better known as CERN) in 1989 while working there. Berners-Lee also defined the HTTP protocol to carry

HTML content, and implemented the first web browser and web server. Already in various implementations, a proposed draft standard for HTML was submitted to the IETF in 1993 [15] and through the 1990's various developments in HTML were handled by the IETF with input from the W3C and software vendors. The details of early HTML development are irrelevant to this project, with the notable points of HTML development being HTML 4.01 [131], XHTML 1.1 [98], and the undergoing development of HTML 5 draft [68].

Basic HTML Layout

HTML defines the content of a page with an Standard Generalized Markup Language (SGML) like syntax. It has some capability to define page layout, but this is generally considered best handled by Cascading Style Sheets (CSS). It uses *tags* and pairs of tags, with each tag enclosed between the characters < and >. Tag pairs make up *elements* that define some property of their contents, with an element closed or *terminated* by a / character. For example, the b element, which makes the content font bold is defined by text . Together the elements in a page make up what is referred to the Document Object Model (DOM), which is a hierarchial representation of the web page.

A page is comprised of an HTML element between a pair of HTML tags, and enclosed within the HTML element are two further elements, the *head* and *body* elements. The head element contains additional document information not including the content, and the body contains page content. The basic structure of an HTML document is shown in figure 2.2.1.

```
<html>
    <head>
        head content
    </head>
    <body>
        page body
    </body>
</html>
```

Figure 2.2: The basic structure of an HTML document

HTML allows insertion of media such as video, sound, and images into HTML documents,

but on its own it does not define interactive content beyond that needed to change the structure of the page displayed.

Security Issues with HTML

Because it is by itself simply a markup language most of the security issues with HTML come from either weaknesses in rendering engines which can be exploited, or from weaknesses or problems with the way inserted media and active content is handled. Active web content is handled by other technologies, the most notable of which is JavaScript.

2.2.2 JavaScript

JavaScript is a client side implementation of the ECMA script programming language originally developed by Netscape[63]. It is implemented in web browsers to allow dynamic and interactive content. While it can be used in an object-oriented fashion, it can also be used in a scripting like fashion, even one function call at a time[43]. This versatility is what enables attacks on web browsers if it is injected or manipulated

AJAX - Asynchronous JavaScript And Xml

AJAX is a term that is used to describe techniques that couple JavaScript and XML for partial manipulation of page content outside of the normal HTTP page-at-a-time operation. The most common example of this is updating small sections of a page from the server without reloading the entire page, which makes the entire process seem far more responsive to users, and use less bandwidth (as only the relevant content need be sent). AJAX is very commonly used in so called *Web 2.0* technologies.

Web 2.0

Web 2.0 is a word that seems to be widely used but that is difficult to define, as its definition seems to change from source to source. A so-called “buzzword, nevertheless it is necessary to define as it is very relevant to the current web, and the security thereof.

The first commonly attributed mention of the term is that of Darcy Dinucci in a 1999 article[41], where Web 2.0 was foreseen as follows:
The web will be understood not as screenfuls of text, and graphics but as a transport mechanism, the ether through which interactivity happens. It will still appear on your computer

screen, transformed by video and other media made possible by ... speedy connection technologies... The web will also appear, in different guises, on your TV set..., your car dashboard..., your cell phone..., and maybe even your microwave.[41]

The term is also commonly used at present to refer to technologies that break the distinction between content producer or publisher, and content consumer. Examples of the types of sites to which the term *Web 2.0* is used today include social networks, AJAX-enabled web applications, and mashups[1].

JavaScript and Security

JavaScript's versatility and ability to manipulate the Document Object Model (DOM), the layout and content of web pages, has also led to the use of it in many types of attacks. Some of the attacks will be discussed in their own section (for example Cross Site Scripting in section 2.5.1, and Cross Site Request Forgery in section 2.5.2). To protect against attacks using JavaScript, the two main security mechanisms that web browsers impose (other than normal software protection measures) are JavaScript sandboxing and same origin policies[63].

JavaScript Sandboxing Sandboxing is the use of techniques whereby code is only allowed to perform a restricted set of operations on a restricted set of items[122, 63]. In web browsers, such techniques are used to restrict code access to other web pages, and (above all) to restrict its access to the underlying operating system. Furthermore, web browsers add a concept of security levels so that the set of allowed operations can be controlled on a more granular level depending upon parameters such as the source of the code. Commonly this will allow more functionality on Intranet sites than Internet sites, and the most privileged functionality is allowed on on files from the local system. Furthermore, there is separation between JavaScript from different domains using same origin policies.

JavaScript Same Origin Policies are used to control access between scripts loaded from different sources. Scripts are commonly considered to be from the same origin if they are included from the same protocol and (sub)domain[111]. Mozilla gives a good table of examples, which we have replicated in figure 2.3.

[1] A *mashup* is a web application (using web 2.0 techniques) that uses content from two or more sources to create a new service not foreseen by the original data providers

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

Figure 2.3: Mozilla's examples on the JavaScript Same Origin Policy [111]

This is intended to prevent scripts reading or writing things to or from other pages loaded in the browser, while still allowing pages from the same origin to interact. Unfortunately these policies are not comprehensive, and many classes of threat cannot be prevented by them. There are ways to successfully violate or circumvent the intent of this policy. The most common methods of circumventing this policy are to include remote JavaScript in elements of a page. For instance by inserting a remote inclusion on http://www.site.com we include a JavaScript element from http://www.attacker.com, now attacker.com can perform any operations that site.com can. Even though the script is not from the same origin, because it is called from site.com it is treated in the same security origin. We detail this in section 2.7.3.

2.2.3 Web Browsers

To handle web content on the client side, a variety of software is used which we will discuss as relevant to this project. The most notable web software type is the web browser, but web browser plugins (discussed next) are also important, as are some other classes of general software which handle web content discussed in section 2.5.6).

A web browser is, at its most basic, an application that is capable of retrieving and rendering HTML documents through HTTP. Modern web browsers, however, contain far more functionality and extensibility, usually supporting additional protocols such as File Transfer Protocol (FTP) and the ability to install "plugins", "addons" or "extensions"(See section 2.5.5) that perform non-core functions.

Web Browser History

Browsers for hypertext documents existed before the WWW web, but we are primarily concerned with web browsers of the modern web. Although initally the only way to view the web

the first browser written in 1991 by Tim Berners-Lee, WorldWideWeb, was quickly followed by Mosaic in 1993, Netscape Navigator in 1994, Internet Explorer in 1995 and Opera in 1996. Mosaic contained most of the common interface elements from modern web browsers, as can be seen in figure 2.4.

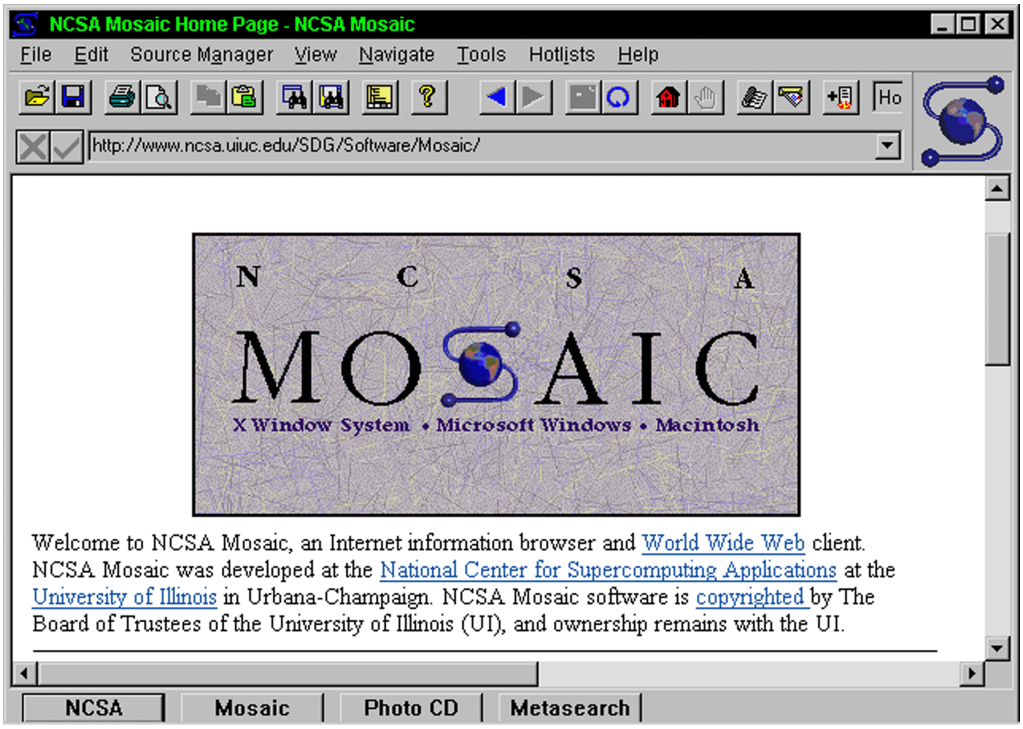

Figure 2.4: A screenshot of the Mosaic 3 web browser. [9]

The successors to these web browsers include the majority of the modern web browser market, with the exception of the WebKit (fork of khtml [1998] circa 2002) based Apple Safari (released 2003) and Google Chrome (released 2008) browsers, which came much later.
The popularity of desktop web browsers at present is, in order of popularity [179, 164, 69]:

1. Microsoft Internet Explorer
2. Mozilla Firefox
3. Google Chrome
4. Apple Safari
5. Opera

These do change in popularity between operating systems. For instance, Internet Explorer is only available on Microsoft Windows, and Safari is far more popular on Apple Macintosh Systems. Also worth noting is that in the ever more important mobile space WebKit based browsers are predominant, with (among others) Apple iOS devices (iPod Touch, iPhone, iPad), Google

Android devices, the latest Blackberry devices, and even some single purpose devices such as the Amazon Kindle Ebook reader running WebKit based browsers.

Web Browsers at present

We cannot discuss all web browsers in use at present, but by covering the common families and versions within those families we can cover the vast majority (over 99%) of web users. We will discuss them in order of popularity as given above; we will then discuss mobile browsers followed by a brief rundown of the security features and architectures on offer by the desktop browsers. All popularity rates are taken from sources for September 2010 [179, 164, 69], but because of different biases all rates differ slightly from source to source.

Internet explorer is available only on Microsoft Windows, and the versions of Internet Explorer in use at present include versions 6, 7, 8, and 9 Beta. They cannot all run on every Windows version however. We summarize compatibilities in table 2.1.

Version	XPSP0	XPSP1	XPSP2	XPSP3	Vista	VistaSP1	VistaSP2	Win7
6	YES	YES	YES	YES				
7			YES	YES	YES	YES	YES	
8			YES	YES	YES	YES	YES	YES
9 beta							YES	YES
Server OS versions (Server 2003, 2008) excluded								

Table 2.1: Internet Explorer Browser and OS/Service Pack Version Compatibility

From the table it can be observed that Internet Explorer 6 is not available on the latest two versions of Windows, and that version 7 is not available on Windows 7 (Internet Explorer 8 is included with Windows 7). Nevertheless as of September 2009 outdated versions of Internet Explorer (6 and 7) have 5-15% and 10-13% share respectively, making up a total market share of around one fifth of the desktop browser market.

Firefox was developed from the Netscape source code which was open-sourced in 1998. Released in 2002, it was originally called first Phoenix, then Firebird, the name changed due to various trademark and name overlaps with other projects until the final name stabilized as Firefox. Currently it is the second most popular desktop web browser with a market share of 20-25%. The versions in use at present are 3.6.x, 3.5.x, and 3.0.x (now end of life after 3.0.19[111]), with

the vast majority of users using versions off the latest 3.6 branch. Version 4 beta is also in limited use, due for release sometime in late 2010.

WebKit [186] is not a browser, but a browser engine that is widely used in a massive variety of applications and web browsers (see sample list[184]). Originally a port of a KDE library, KHTML, WebKit was developed intially for Apple's Safari browser[184], but is now used in a far larger variety of applications and contributed to by many large oprganizations [185]. The most important WebKit based applications for this project are Google Chrome/Chromium, Apple Safari, and many mobile browsers, we will discuss these ass appropriate below.

This widespread adoption of the WebKit engine means that on desktop web browsers webkit has a market share of around 10-15%.

Google Chrome was released in late 2008. Developed by Google, it is a WebKit based web browser the code for which was also open sourced in a project called Chromium[59]. Chromium's source code is from a variety of sources using many different licenses [19]. For instance, Google developed the JavaScript Engine (v8). Chrome and Chromium are based upon the same main chromium codebase, but Chrome has some additional non-open source features that Chromium does not, such as an automatic updater, integrated Flash player, PDF viewer, and additional media codec support[18]. For the purposes of our work, however, we shall refer to the Google Chrome browser and not Chromium builds. With major version releases every few months, Chrome is the third most popular browser at present. Due to the automatic silent updating most users are using versions from the last two major releases, at present[2] versions 5 or 6, although the public *beta* version 7 and the development (or *dev*) version 8 releases are in use by a few testers.

Safari was released to the public in January 2003[8], originally as a browser for Mac OS X (becoming the default web browser with Mac OS X 10.3 or *Tiger*), but was later also released for Windows in 2007 [7]. Safari has a small market share of around 5% on desktop devices. Apple's iOS which is used on the iPhone, iPad, and iPod Touch devices uses a version of Safari as its primary web browser (discussed in more detail below).

Opera is the browser with the smallest share that we shall discuss, with a market share of around 2%. It has a history that goes back to 1995, making it one of the older contenders. It

[2] As of September 2010

retains a small market share, but has some features that make it worthy of note. Although not a feature, the property which many use Opera for is that fact that it is a fully usable web browser, but because it remains separate from the main browsers in codebase and is small in market share, it is thus less likely to be targeted by attackers.

Mobile Devices at present have a large variety of software arrangements, however in the more functional and power so-called *smart* devices the variety is significantly less, especially in web browsers. The majority (over 99% in many cases eg[138]) of the smart devices browsing the web consist of devices running Apple's iOS, Google's Android, or Blackberry OS. All devices running iOS or android, and the latest Blackberry devices all use WebKit based browsers. The browsers do differ significantly in features and layout (and things like JavaScript engines) but with the same family of rendering engine making the mobile browser space at present quite homogenous when compared with the desktop space.

2.3 Internet Content Adaptation Protocol (ICAP)

The Internet Content Adaptation Protocol (ICAP), is a standard for content modification servers that was developed by Network Appliances in 1999 and has been used in the decade since. ICAP is defined in RFC 3507[46] and the ICAP Forum[75] provides some technical discussion with industry members and other stakeholders.

2.3.1 General ICAP Overview

The Internet Content Adaptation Protocol (ICAP), as its name suggests, is a protocol designed for modifying Internet content. ICAP is an HTTP-like protocol discussed in the IETF informational rfc 3507[46]. It is designed to facilitate the modification of content that passes through proxy servers. Introduced in 1999, it is a standard protocol for communication between proxy servers and callout (separate content modification) servers. ICAP is able to receive and modify both HTTP requests and HTTP responses. Our descriptions of ICAP operation are all derived from RFC 3507[46]. It is worth noting that ICAP can only inspect what the icap-client (proxy) can see. Generally this means it cannot inspect any encrypted traffic (we will discuss this later in sections 2.6.3 and 7.2.1).

2.3.2 ICAP Overview

The architecture for an ICAP system is essentially the same as for an HTTP proxy, but with an additional server (the ICAP server) that the proxy forwards both requests and responses to so it can check if modification is needed. The proxy server is called the *ICAP client*, and the content modification server is designated the *ICAP server*.

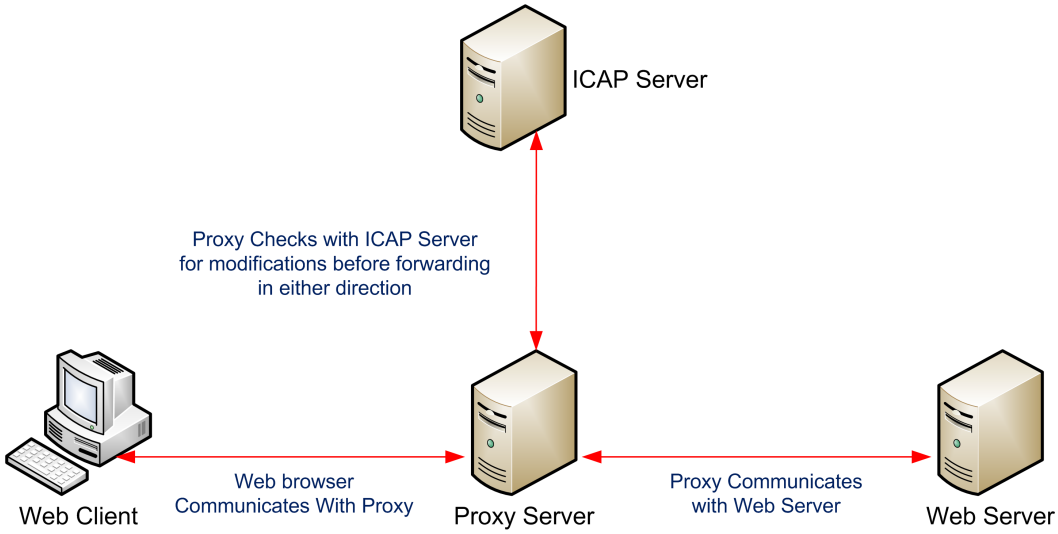

Figure 2.5: A Proxied connection utilizing ICAP

General ICAP operation

An ICAP server performs two main operations: HTTP request (see section A.1.3) modification, and HTTP response (see section A.1.4) modification. By monitoring and modifying both directions, full supervision and control over the HTTP stream can be attained. Request and Response modification are performed by two ICAP operational modes designated REQMOD and RESPMOD; these shall be discussed later, but first we shall outline general ICAP operation.

ICAP Operation Steps

The general ICAP operation is as follows for content alteration. Figure 2.6 has the steps labeled, and these are explained in brief immediately following.

We shall use the same terminology used for HTTP in section A.1.3 for ICAP regarding requests and responses: *request* denotes communications from the ICAP client (proxy) to the icap server, and *response* denotes communications from the ICAP server to the ICAP client directly caused by an ICAP request.

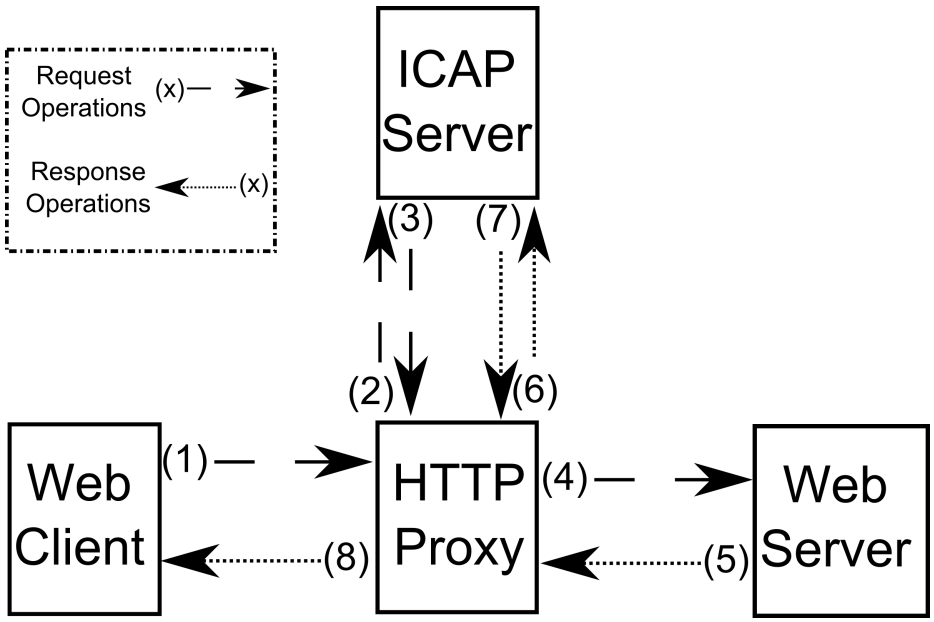

Figure 2.6: Numbered steps in a proxied connection utilizing ICAP

1. Client system requests the content from the proxy.
2. Proxy forwards request to the ICAP server (reqmode)
3. ICAP server responds to proxy with either modified request or an OK to continue unaltered.
4. Proxy sends the request (or modified request) to the web server itself
5. Web server responds to the proxy as appropriate
6. Proxy forwards the received response to the ICAP server
7. ICAP server responds to the proxy with either modified response, or an OK to continue unaltered
8. Proxy responds to client system with either the response from the web server or the modified response from the ICAP server

2.3.3 ICAP Operation

ICAP Request Format

ICAP is a text-based protocol like HTTP, and looks very similar to the HTTP protocol, although it is much simpler. ICAP requests are in the style of an HTTP request chain (section A.1.4), but with a slightly different makeup. An *ICAP request chain* is the series of communications that is sent from the client to the server. An ICAP response chain contains the following separated by newlines (CRLF) with the body separated from the headers by a further blank line:

1. Start/Request line
2. ICAP Request Headers
3. ICAP Body

Start/Request Line

The first line in an ICAP request is very much like the first line in an HTTP request, the status line is of the form:

icap-method <space> icap-uri <space> icap-version <CRLF>

The ICAP-method parameter is one of three ICAP methods, namely: REQMOD, RESPMOD, or OPTIONS. For this project we are only directly concerned with the REQMOD and RESPMOD methods. REQMOD indicates that the ICAP server should perform *request modification*, that is that the rest of the message will contain an HTTP requests that the server can act upon. Conversely RESPMOD indicates that the ICAP server should perform *response modification* upon the HTTP response contained within the ICAP body of the message.

The ICAP-URI parameter is used to hold the address for the ICAP server. It may also be given a path and query, but these are not normally used except to allow conceptual separation of request and response modification. In the case of an ICAP server however the scheme is icap://, and the default port is 1344.

For a request example, if the ICAP server is at the ipaddress 192.168.1.1, the relative location of the request response processing is /req_mod the the ICAP-URI will be:

icap://192.168.1.1/req_mod/

The icap-version parameter is used to indicate what ICAP version the requesting ICAP client (proxy) supports, this will always be 1.0 at present, as it is the only version.

ICAP Request Headers

ICAP Request headers follow the same format as HTTP headers, but differ in content. The most important part of this for the purposes of this work is the encapsulated HTTP header, which indicates details regarding any HTTP content carried in the ICAP-body section. In this project however neither this nor any other ICAP headers were ever directly modified, and were left to be automatically handled by the ICAP client (the proxy server).

ICAP Body

The ICAP body is simply the full HTTP request, request headers, and request body repeated after an empty line.

Example ICAP Request

The example below illustrates the makeup of an ICAP request, and is modified from one in RFC 3507 [46] section 4.8.3.

```
REQMOD icap://icap-server.net/server ICAP/1.0
Host: icap-server.net
Encapsulated: req-hdr=0, null-body=170

GET / HTTP/1.1
Host: www.origin-server.com
Accept: text/html, text/plain
Accept-Encoding: compress
Cookie: ff39fk3jur@4ii0e02i
```

Figure 2.7: An example ICAP request (modified from RFC3507[46])

ICAP responses follow a similar syntax.

ICAP Response Format

The ICAP response format is very much the same as the HTTP response format according to the definitions in the RFC's[3507]. An ICAP response chain contains the following separated by newlines (CRLF) with the body separated from the headers by a further blank line:

1. Response line
2. ICAP Response Headers
3. ICAP Response Body

Response Line

The first line in an ICAP request is very much like the first line in an HTTP response, of the form:

ICAP-version <**space**> **status-code** <**space**> **status-description** <**CRLF**>

The ICAP-version parameter will always contain *ICAP/1.0*, as this is the only ICAP version.

The Status-Code and Status-Description parameters act the same as in HTTP – they handle not only the content but also error handling. The status codes follow the HTTP pattern of 3 digit numeric numbers from the protocol specification, where the first number denotes which class of status. ICAP status codes also follow the HTTP pattern, with the exception of the 300 (redirection) codes, which are not used.

There are four classes of status code used in the ICAP specification:

- 1xx Informational
- 2xx Success
- 4xx Client Error
- 5xx Server error

While there are numerous status codes in the specification, for our purposes only a few are relevant, namely “200 - OK”, and “204 - No Modification”.

The response status 200 - OK indicates success for the request. In the context of ICAP this means that MODIFIED content follows in the ICAP body. This may be either a modified version of the original request or response, or in the case of request modification the ICAP server may reply with a full HTTP response directly, thus the request is never sent to the web server.

The response status 204 - No Modification indicates that the ICAP client need not modify the content but may send it on unmodified. **Note that in this case the ICAP body is empty, it is assumed that the ICAP client kept a temporary copy.**

ICAP Request Headers

ICAP response headers are very similar to HTTP response headers. They do however define content encapsulation exactly the same way as in ICAP request headers, however in this project we never directly access these and leave them to be automatically handled by the ICAP server.

ICAP Body

The ICAP body is simply the full HTTP response, HTTP response headers, and HTTP response body repeated after an empty line.

2.3.4 Considerations for ICAP Use

When compared with competing technologies ICAP has some relative advantages and disadvantages that must be taken into account when planning a deployment. The considerations surrounding ICAP are context dependent.

The use of a remote server for content modification means that ICAP is remote to the proxy infrastructure, and removes much of the dependence upon specific proxy implementations. When combined with the statelessness of ICAP, load balancing is also facilitated across multiple ICAP servers.

The requirement for the use of proxies does, however, have some drawbacks. Due to the way they function, all traffic passing through proxies is visible to them, which has potential privacy concerns. The proxies also become points of failure, as they are central to operation: if they fail then any clients using them cease to have web access. Furthermore, due to the versatile nature of ICAP it can also be used for nefarious purposes that are either subversive, or undesirable from a user perspective.

Many related issues for content modification and the effect of its use upon the integrity of web content are discussed in [53]. Of the measures discussed to maintain trust and integrity, we use the notification and non-blocking properties. The notification property means that users will be notified of modifications, and the reason for them, while the non-blocking property is implemented so that users can still access non-modified content after the notification if they wish.

2.3.5 Alternatives to ICAP

ICAP is not the only content modification technology in use or proposed, however. Alternatives include similar callout server[3] based approaches, as well as proxy level approaches. While approaches based upon client based software are not directly comparable as a means of content modification in the context of this work (as they do not use a single web gateway for content

[3] A callout server is a server that some sort of client calls out to to request some information

modification), they are important in the context of security measures however, so will be discussed briefly.

The OPES Callout Protocol (OCP) [4] is a protocol proposed by the Open Pluggable Edge Services working group for content modification via callout servers. Discussed in [139] it has also been proposed for use as ICAP 2.0. At present, implementations are application specific, and no widespread implementations are available.

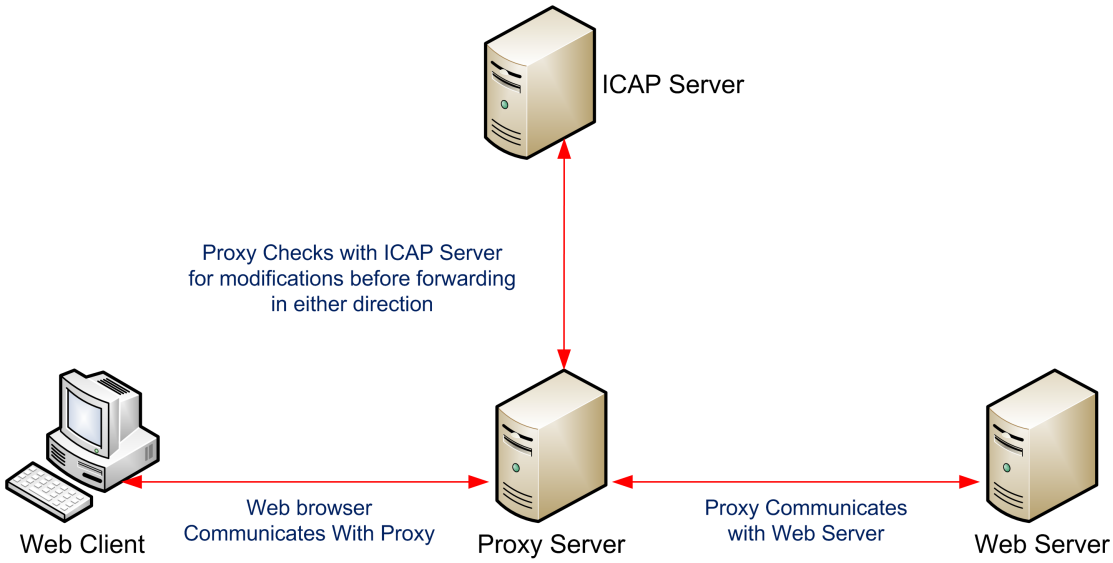

Figure 2.8: In an ICAP or OCP based security system, the ICAP Server handles the security operations.

ECAP (Not an acronym) has a name similar to ICAP, but ECAP is not discussed in an RFC like ICAP. Furthermore it achieves content modification via a different approach.

ECAP is at a very early development stage (version 0.02), and is not yet widely supported or documented. The E-cap.org website describes eCAP as follows:
eCAP is a software interface that allows a network application, such as an HTTP proxy or an ICAP server, to outsource content analysis and adaptation to a loadable module. For each applicable protocol message being processed, an eCAP-enabled application supplies the message details to the adaptation module and gets back an adapted message or a "not interested" response. These exchanges often include message bodies.
If you are familiar with the ICAP protocol (RFC 3507), then you may think of eCAP as an "embedded ICAP", where network interactions with an ICAP server are replaced with function calls

[4] OPES = Open Pluggable Edge Services working group

to an adaptation module.[169]

Proprietary proxies and modified existing proxies are among the the most flexible methods for content modification, but also the most labour-intensive. Setting up and modifying the behaviour of such proxies via source code modification is a very intensive and time consuming process, requiring a software restart. Building a proxy from scratch also requires a large investment.

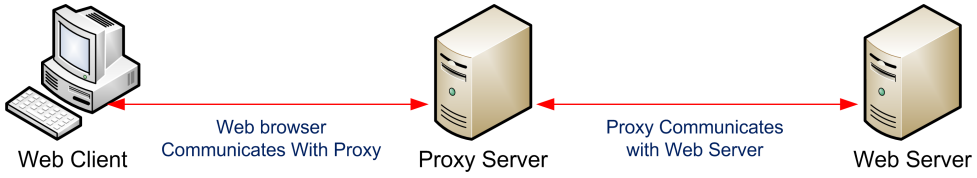

Figure 2.9: A proprietary web security proxy handles all content sanitation itself

Proxies which have been modified or built to facilitate content modification can perform the modifications natively with no use of a callout protocol either locally or remotely. By developing a product from the ground up, intellectual property rights to the full product remain with the developing organization, and unlike modifying open source products there are not normally any required disclosure issues. For many commercial secure web gateways, however, while documentation on their claimed functionality abounds, little detail is publicly disclosed concerning specific details of operation.

Clientside Content Modification Mechanisms may be functionally similar to proxy based methods, but are similar only in that they modify content and not in the mechanism by which they achieve it. Content is modified on the client itself via the use of a local proxy or via software that hooks incoming and outgoing network traffic and modifies it on the fly. While central management is possible for these systems in some cases, normally it is more complicated since integrity of the client is required for full functionality.

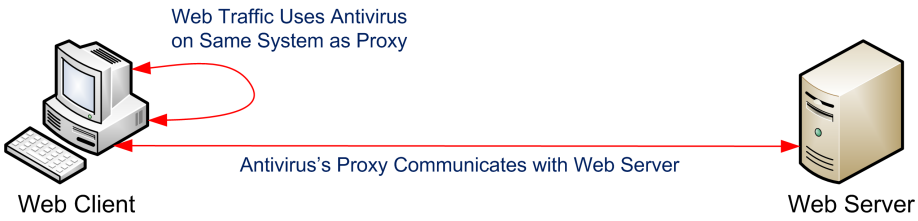

Figure 2.10: Client Based Web Security proxies through a local proxy

2.3.6 Example Uses Of ICAP

ICAP is implemented in many products and devices, and also is detailed as being suitable for others. Among the things for which ICAP is, or is detailed as being suitable for are:

- Advertising insertion
- Content censorship
- Content transformation (e.g. mobile device proxies [181])
- Automated translation (e.g. redirection through translator[40])
- Virus scanning
- User message insertion [20].

Current Security Implementations For ICAP

The utilisation of ICAP to modify page content has been investigated for filtering of objectionable content, and also for other content transformation situations such as content classification[55] and the modification of pages to reformat them for mobile devices[181]. This project investigated the use of ICAP for security, and there are numerous security publications and security products that use ICAP for their operation[5]. At the time of writing, the products listed on the ICAP-forum webpage as implementing ICAP [76] included multiple commercial security products implementing ICAP servers. These include antivirus scanners, censorship servers, dataloss prevention and Secure Web Gateways (discussed in 2.4.1). Security products not listed by the ICAP-forum also claim to utilize ICAP (such as the DrWeb Antivirus for Internet Gateways [183]). Thus ICAP is used both in literature and in commercial products for similar purposes to those in this work.

2.4 Related Technologies, Protocols, and Services

2.4.1 Secure Web Gateways

A secure web gateway is a centrally managed system that monitors network perimeter points and is designed to protect clients from web security risks (such as malware). It is designed to prevent both the entry of external threats (such as malware) into the network, as well as the exfiltration

[5]In addition, mentions were found of an open source proxy that used ICAP and Clam Antivirus to scan HTTP content (httpav), but it has since disappeared, and is presumably abandoned.

of data by internal threats. They generally function as proprietary filters or proxies, however some commercial implementations support the ICAP protocol [132].

The normal functionality of a SWG includes URL filtering and anti-malware, although other functionality may be included. For a discussion of the enterprise products and the enterprise market see[132].

2.4.2 Twitter

Twitter[173] is a web based *microblogging* service that was originally designed to allow messages of only 140 characters, to fit within the constraints of cellular messages (SMS). It is widely used on desktop web browsers, desktop applications, and mobile devices. Because messages are so short it has a very quick response time to events, with some message trends lasting only minutes, trends in the messages that flow through it are very indicative of the things that users are discussing. Twitter is very commonly used to publicise things quickly, but also it commonly used to share links, although to fit within the length constraint most of these are shortened URLs.

2.4.3 Short URL Services

Short URL services offer redirection service that takes full URLs, and maps them to shorter URLs at the domain of the URL shortening server. While they are run privately, and many companies run their own, many are also available for public use. They use HTTP redirects, and often track statistics.

Some of the most well known public URL shorteners include:

- bit.ly
- tr.im
- goo.gl
- ow.ly
- is.gd
- tinyurl.com

They may each be used by going to their URL and requesting a shortened link. For example, instead of:

http://www.alongurl.com/folder/subdirectory/anotherfolder/filename.htm

which is 70 characters long, the user can post a message containing:

bit.ly/c9Lgcz

which is only 13 characters long.

Two notable examples of companies using their own servers for URL shortnening include the New York Times with nyti.ms, and Youtube with youtu.be.

Mechanisms for Short URLs

URL shorteners look up the requested short URL key (the part after the /), and redirect the web browser to the full URL. Although they could use other mechanisms, most use HTTP 301 (moved) redirects, but some use 302 (found) or 303 (see other) redirects. The exact mechanism is not relevant to our work.

Additionally, the server that handles short URL processing can gather and store information regarding the use of the links. This includes basic things, such as a count of the number of times a particular short URL has been used, the IP address of the request, and additionally information from the HTTP headers of the requesting browser. The gathered analytic information is usually publicly visible too, enabling a party who is not in control of a site to track some usage of it via monitoring short URLs that point to it. The statistics can contain details such as a count of accesses, referrer details, and country location from IP addresses. An example statistics page from bit.ly is shown in fig 2.11.

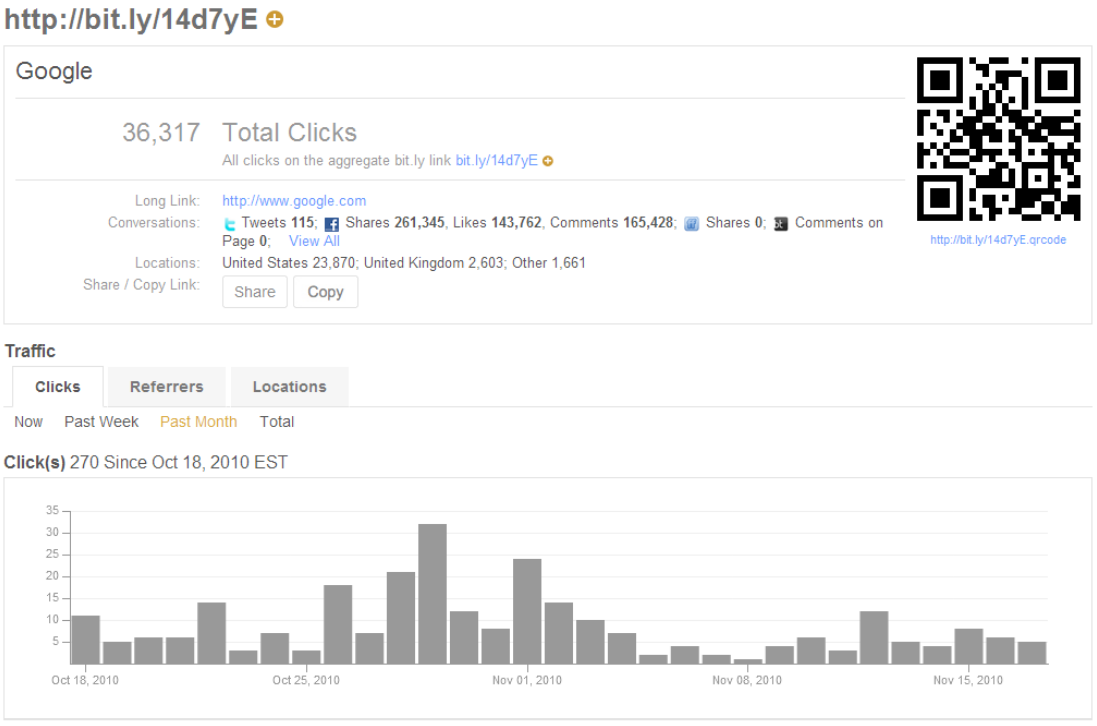

Figure 2.11: Bit.ly's short URL statistics page

Security Risks surrounding Shortened URLs

Shortened URLs present a few types of security risk to users due to the way they operate. Because URL redirection services break the standard one-link-to-one-resource hyperlink model,

the end user is potentially exposed to a greater security threat than would occur with a direct link. Short URLs are non-transparent, that is, the destination of the link cannot be seen by end users before clicking it, and also third party as neither the person requesting the link, nor the user controls the short URL server. Thus, users cannot know where they are going, or even if they are going where intended – this has been abused by phishing attackers and malware distributors alike. "Link rot" is also a potential risk, as if the short URL service ceases to function, then all links going through it will cease to operate too, even if their end destination is still available.

System Virtualization and Virtual Machines

In this project, system virtualization was used in the client honeypot components (discussed in section 2.4.3 next) as the oversight and control afforded allows guest states to be monitored, captured, and restored quickly.

System virtualization is used to abstract software away from the hardware upon which it runs. This performed with the use of a Virtual Machine Monitor (VMM) which provides software with the same interface (inputs, outputs, and operations) that physical hardware normally provides. The use of a VMM means that the physical hardware state is no longer relevant multiple operating systems to be run on the same hardware, each inside a separate *virtual machine* (VM). A virtual machine represents a logical machine that runs upon the VMM, and may have an operating system installed upon it (called a guest). Each guest will perceive that is has complete control over the system, whereas in reality the VMM has full oversight and control over all operations of the virtual machine. A VMM may run multiple virtual machines, and thus may have multiple guest operating systems running on the same hardware.

Figure 2.12 gives the VMM architecture as used in this project, a so called *Type II* VMM, where the VMM is installed alongside an operating system on the host physical machine, and uses the operating system on the host machine to handle Input/Output (IO).

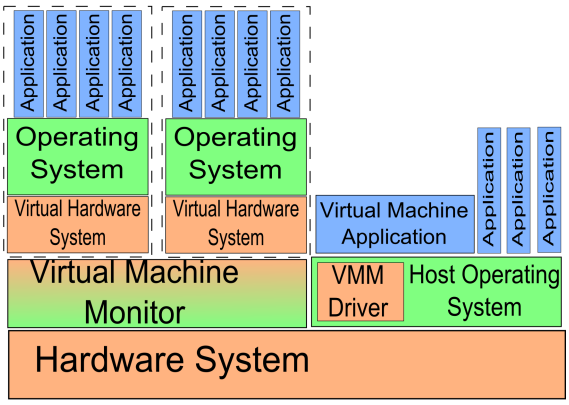

Figure 2.12: The type II virtual machine architecture, as used in this research (VMware Server) [136]

Client Honeypots

A honeypot is a system that is set up to either deliberately attract attackers and then observe behaviour, or to distract them from critical systems. Normally a network honeypot is a server that has open ports and waits for attacks. A *client honeypot*, is a client system that performs actions as a client and waits for attacks. Client honeypots comer in two main types: *High Interaction* (HI), and *Low Interaction* (LI). A Low Interaction Client Honeypot (LICH) is a honeypot that provides an emulated environment for attacks, wheras a High Interaction Client Honeypot (HICH) runs actual software that a client could be expected to run. We only briefly discuss client honeypots as they relate to our research 2.7.3, for more detail see: [31] [150]

2.5 Relevant Threats

Like any untrusted network, the Internet has many different threats that it introduces, however those of interest to this project are those that affect clients and may be mitigated[6]. Many of the negative effects on clients can be prevented or detected by monitoring the client or network connection, including most attacks where arbitrary code or commands are run on or by the client.

2.5.1 Cross Site Scripting

Cross Site Scripting, or XSS is a form of code injection attack that abuses the the trust model and mixed data/code/presentation nature of HTML to do things that the website creator did not intend. Because the same origin policy of JavaScript treats all code originating at the same

[6]There also exist many threats surrounding servers, but they are not of interest to this work.

subdomain at the same trust level the code displayed, if an attacker can get the web server to display something on the page, then it is run on clients exactly as if the server itself requested it. A cross site scripting attack is when an attacker manages to inject some content, code, or data so that the web server displays it in a page sent from the server and is run on the client. For more detailed discussion of XSS attacks see: [119, 191, 188, 178, 77]

There are three main types of XSS: reflected (also called non-persistent), stored (also called persistent), and Document Object Model based. While it intially appears a trivial threat, many other types of attacks can be executed with the help of XSS, including page defacement, session stealing, and phishing (among others).

Reflected XSS attacks abuse pages which use some form of server side scripts to display user input back to a user. Common examples of this include site search, which will often repeat the search terms in a message on the results page. For example, if searching for "fish tanks", the results page may contain the line:

your search for "fish tanks" returned 45 results

So, if an attacker was to use JavaScript code as the search terms it would be displayed and run on the results page. This does not generally allow an attacker to target another user, however. To target another user, there needs to be some way that a user can visit the results page directly; often this is done via *query strings* in the URL. A query string is a part of a URL that defines a parameter passed to server side scripts. So, to search a hypothetical site for fish tank accessories the full URL may be:

http://www.site.com/search.php?search=fish+tank+accessories

An attacker may enter malicious code instead of search terms, thus, if they can get a user to visit the URL in question then the code will be run. When combined with short URLs this becomes a far easier task to accomplish, as the user cannot see anything about the link destination before they click it (and even if they did they are likely to see the domain and assume all is safe).

Persistent and DOM-based XSS Persistent XSS is similar to reflected XSS, where user input is injected into site generated code, but it does not depend upon information passed in by the user's browser (such as query strings). Common examples of stored XSS include user comments or forum postings where user input is not adequately screened or sanitized to stop user input running on other users' browsers.

Document Object Model (DOM) based XSS is similar to reflected and persistent XSS, but

differs in the source of the data that the attacker manipulates. In a DOM based XSS attack, the attacker modifies an aspect of the page that appears in the output, but not a standard aspect like a query string. This can include manipulating the referrer, page anchor target, or other technical aspects of the page model.

These forms of XSS are not relevant to the work of this project, as countermeasures that are not server based need to be quite sophisticated. Conversely, countering reflected XSS in the most basic form simply involves checking whether aspects of the request that are reflected in the output contain active content.

2.5.2 Cross Site Request Forgery

Cross site request forgery (XSRF) attacks are similar to reflected XSS attacks in that they involve manipulating the client browser into doing something unintended, however where XSS aims to manipulate server output, XSRF attacks aim to manipulate server operations. This is largely caused by the way cookies work to maintain login state on websites - if a user is logged in to a site, then every request sent to that site will contain the cookie. This can be abused if the user can be made to send a request for the server to do something, such as message a user. Consider the following example:

If the way to post a forum message is to send a POST request to webforums.com/postmessage.php containing the message while logged in, if JavaScript attacker.com forces the user's browser to send a spam message to webforums.com/postmessage.php, this will then be posted as that visitor if they are logged in.

2.5.3 Malware downloads

Malware, or malicious software is software that performs some action the user does not desire or has no control over. This can include any operation the system can normally perform, and also operations the malware author or controller may desire. This can lead to data loss, personal information disclosure, the sending of spam, or other undesired consequences.

Few users intentionally infect their machine, but many are infected nonetheless. The vectors malware uses for entry systems are quite varied, but some common forms include email attachments, *trojan horses* (unwanted software that is hiding in legitimate-looking software), worms (automatically propagating through the network – not in scope of this work), or drive-by-downloads (which we discuss next). In many cases is it very difficult for a user to correctly identify malware as dangerous before they download it.

2.5.4 Drive By Downloads

Drive-by-download (dbd) is a form of malware propagation that makes use of client software vulnerabilities to download and install malware on a machine. The user need only visit a website containing (or including) attack code and a weakness in a software component on their system is exploited to infect their system. Commonly this will involve buffer-overflow exploits in the browser or plugins, but it infection may also occur via other mechanisms such as improper URI-handlers. Exploit packs, server side software designed to automatically fingerprint software on the visitor and deploy the appropriate attack, have made this form of attack far more common.

Drive by downloads are particularly dangerous due to the lack of requirements for user intervention and ability to infect if only a single software component is vulnerable. Some discussions of their prevalence, mechanisms, and countermeasures can be found in [182, 108, 147, 106, 129, 128, 152, 101, 104, 45, 44, 102, 103, 91], and further discussion follows in section 2.7.3.

2.5.5 Important Web Browser Addins, plugins, or extensions

Browser plugins have become particularly important as an attack surface [5, 17, 56], because not only do they often not contain as many protection measures as the browsers themselves, but as they can call external executables and libraries it is possible to affect multiple browsers through the same plugin. The most attacked recently include Adobe's PDF and Flash plugins, and Oracle's (formerly Sun's) Java[94, 103], although many others are important, such as media players. Almost every type of attack at present is a file format mishandling vulnerability.

The important classes of browser addons and extensions include those for handling Rich Media, Multimedia, Scripting, and additional types of document file formats.

Rich Media Plugins display their own non-HTML dynamic content media format inside sections of the web browser's display of the page. These include the very ubiquitous Adobe Flash and Microsoft Silverlight, as well as the less common Adobe Shockwave. Although they display web content, they run separate executables that until recently were not sandboxed as effectively as web browsers.

Multimedia Plugins are similar to dynamic media plugins, but they are more specific to displaying audio or video content. The common multimedia plugins include Apple Quicktime, Microsoft Windows Media Player, and Realplayer, although there are many others available.

These normally render the content with helper software, and in most cases have associated URI-handlers that can be used to call the external software to display the content as well.

Document Plugins are similar to the rich media plugins, but are used to display particular document file formats. This is used to allow the convenience of viewing the documents without loading external software. The most common example is the Adobe PDF viewer, but Microsoft Office formats are also often viewed in a web browser. These plugins try to bring the full range of their component formats into the web browser setting, including nacros and scripting.

Scripting or Programmatic Content Plugins are used to add additional programmatic functionality that web sites may use. In contrast to the other plugin types these are more flexible and less constrained in what they do. Instead of performing a constrained set of operations, these can perform the full range of operations that their security settings allow. Common examples of this include Java Applets, which are a form of Java programs that run inside web browsers, but web browsers may also support full Java programs through the use of the external Java runtime.

2.5.6 Other web aware software

Software such as media players, email clients, and office software is also a risk to drive by downloads, as if it is vulnerable and can load remote code (for instance a file via a URI-handler), then exploitation can occur.

2.6 Protecting Clients

Protecting clients against threats that the web presents is a large area of research, with many different proposed solutions. There are many different measures proposed to deal with specific threats that systems may use. Systems that attempt to mitigate the risks use a few different architectures and may be divided into the following main classes (or combinations of them): hardening client software, client based detection and mitigation measures, and centralised or network point of entry based solutions. The architectures are discussed first, followed by discussion of some specific mitigation measures.

2.6.1 Hardening client software

The most prevalent piece of web software is the web browser, and indeed much of the recent development in web security over the past decade or so has centred around making web browsers more difficult to exploit.

Web Browsers and Security

Web browsers were initially written as normal software, simply requesting and rendering a file format. Inherently, however, the Web is untrusted, and as the web and the associated threats have evolved different approaches have been applied to their mitigation. Frei et al[56] note a rising prevalence of web browser attacks, and discuss web browser security in depth.

Unlike some security domains, the web is inherently untrustworthy. No Internet facing website should be considered fully trustworthy, especially those that are unencrypted and contain large amounts of active content. As attackers have broadened their attack vectors to not only infect from websites commonly considered risky or untrustworthy (such as pornographic or *warez* sites), but to also infect users of legitimate or otherwise trustworthy sites. This is usually accomplished by methods of remote code inclusion such as remote advertising, but also through compromising the website itself and injecting content on it.

Browsers have also started to natively support memory protection measures intended to prevent or detect memory corruption exploits such as buffer overflows, and also measures that make the crafting of successful repeatable exploits much more difficult. This has been assisted by operating system development improving the available protections also. For instance, on Windows buffer overflow prevention measures include Data Execution Protection (DEP), while detection measures include stack cookies (/gs protection) and exploit complication measures include Address Space Layout Randomization (ASLR)[7]. The technical details of such protections are not relevant to this work, and protections change from operating system to operating system and version to version, so this list is not comprehensive. However, the acceptance of browser vendors that their products will be attacked and will fail is an incredibly important shift of philosophy. While these measures make attacks more difficult, they do not completely prevent successful attacks, nor protect older versions of the software.

Outdated versions of Internet Explorer are one of the biggest web browser security issues. Furthermore, many of these outdated browser versions require old version operating systems

[7] Address Space Layout Randomization - This changes the address of absolute memory addresses, to make it more difficult to write static exploits that jump to run code in a known address.

that are less of a challenge to exploit than later versions (see table 2.1). Thus, around one in five desktop browser users are using insecure versions of Internet Explorer, and are far more likely to get infected with malicious software that users of later versions [104, 103, 56]. Many of these users will have automatic updating disabled, as early versions of Windows XP did not enable it by default, or because of corporate patch or version management policies. This is exacerbated by the non-standard rendering of some HTML pages in Internet Explorer 6, motivating some organizations to keep using version 6 for legacy web application support without rewriting the application. This is becoming a more difficult position to maintain, as many websites have stopped supporting version 6. Among the most notable organizations to have done this is Google [156].

Other web browsers such as Firefox, Safari, Chrome and Opera have vulnerabilities of their own that are important [5, 17], but because these have less market share they are less commonly targeted.

With more recent web browsers have come not only security patches, but changes in architecture and security model in an attempt to deal with security threats online. Most notable among these at present are sandboxing (which has been extended beyond just JavaScript), and automatic updating, which now occurs on most browsers. The mainstream browsers started to apply such techniques in the mid to late 2000's, notably Internet Explorer 7's *protected mode*[157] which attempted to sandbox the browser process through applying Mandatory Access Control (MAC), and later Google Chrome with process separation between tabs and the multiple layers and separation of sandboxes for different components (such as rendering processes)[42] – See figure 2.13.

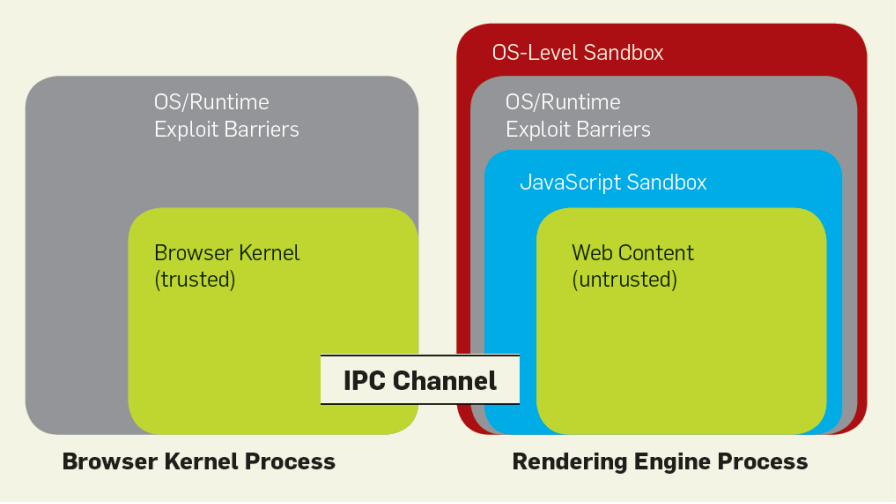

Figure 2.13: Chrome's multi-layered sandboxing, from [134]

The most common attack vector used to attack web users is thus outdated or unpatched software which contains unfixed vulnerabilites [104, 103, 56], especially if the vulnerabiites are publicly known and in use (also referred to as *in the wild*). This is relevant not only to web browsers, but also to software which uses web browser components or to plugins in web browsers.

Sandboxing and memory protection measures made it more difficult for attackers to target the web browser itself, so they migrated their attacks to target the browser plugins, which were not commonly sandboxed. As a response more recently browser architectures are beginning to either sandbox browser plugins, or to include sandboxed versions for particularly targeted ones[48, 56]. This approach is augmented with warnings to users of outdated plugins, for instance figure 2.6.1.

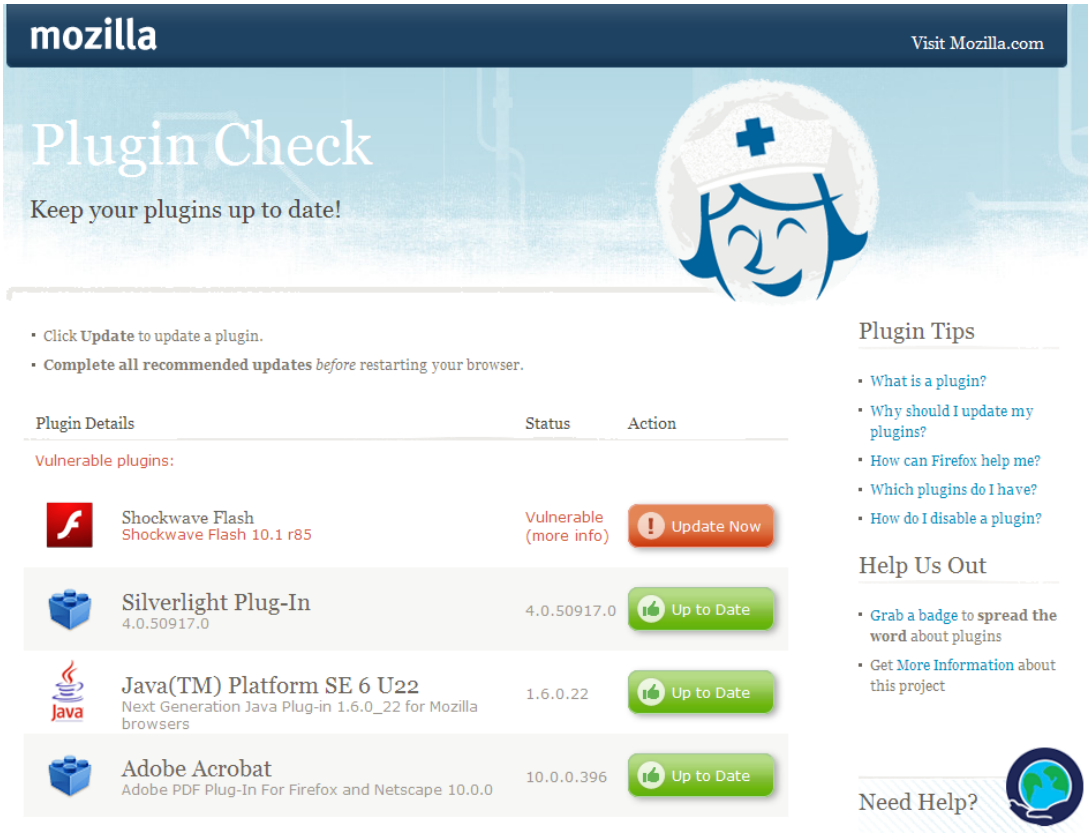

Figure 2.14: The Firefox Plugin version check page [110]

A summary of the security features for the notable browser versions in use (historical, or of approximately 1% use or more) is given in table 2.2. Note that some browser security features and their threats are yet to be discussed.

Browser Version	**Released 20xx**	**Sandboxing**	**Automatic Updates**	**Reputation Warnings**	**Popup Blocking**	**Private Browsing**	**XSS Filter**	**Exploit Mitig Enabled**[124]
Int Explo 6	01	No	Non default OS Setting	None	Some Vers	None	No	Non Default
Int Explo 7	06	Security Lvls	Non default OS Setting	No	Yes	No	No	Non Default
Int Explo 8	09	Security Lvls	OS Setting	Yes	Yes	Yes	Yes	Yes
Int Explo 9	TBA	Tabs	Yes (Notify)	Yes	Yes	Yes	Yes	Yes
Firefox 2	06	No	No (Notify)	Yes	Yes	No	No	No
Firefox 3.0	08	No	Yes (Notify)	Yes	Yes	Yes	No	Partial
Firefox 3.5	08	No	Yes (Notify)	Yes	Yes	Yes	No	Partial
Firefox 3.6	08	Plugin (Partial) [109]	Yes (Auto)	Yes	Yes	Yes	No	Partial
Opera 9 [120]	06	Unknown	Yes (Notify)	Yes	Yes	No	No	No
Opera 10 [120]	09	Unknown	Yes (Auto)	Yes	Yes	Yes	No	Partial
Chrome 3,4	09	Tab	Yes (Auto)	Yes	Yes	Yes	Some	Partial
Chrome 5,6,7	10	Tab	Yes (Auto)	Yes	Yes	Yes	Yes	Yes
Chrome 7, 8	10	Some Plugins	Yes (Auto)	Yes	Yes	Yes	Yes	Yes
Safari 3	07	Unknown	Yes (Notify)	Yes	Yes	Yes	No	No
Safari 4	09	Unknown	Yes (Notify)	Yes	Yes	Yes	No	No
Safari 5	10	Unknown	Yes (Notify)	Yes	Yes	Yes	Yes	Partial

Table 2.2: Browser security feature support (Sept 2010)

Note: Sources for data on this table comprise around 40 sources; as this data is easily publicly available, source detail has been omitted for brevity.

2.6.2 Client based detection and mitigation Approaches

Client based solutions to preventing web threats (other than hardening software) come in a few main forms and include web browser addons, antivirus and anti-malware software, and web security software and firewalls. Here, their general protection models are discussed and some specific protection measures are discussed afterwards in section 2.7. Note that often these models are integrated together in software packages or suites.

Web Browser Addons are software components that run inside the web browser, and thus can process information at the web browser endpoint. Because they are logically located at the web browser layer, they can process any and all data as the web browser receives or sends it, irrespective of what occurs at any other layer such as network encryption. While offering good flexibility, web browser addons are difficult to centrally manage, and can become problematic for system performance and web browser usability if too many are used.

Antivirus and Anti-Malware/Anti-Spyware Software consist of software installed on systems to detect, prevent, remove, and mitigate the effects of, unwanted or dangerous software. The detection of unwanted software occurs through two mechanisms, signature based and heuristic. Signature based approaches scan files and Input/Output sources for characteristic byte sets or patterns that are contained in a signature database. Heuristic approaches examine the operation of software both before execution (through methods such as decompilation or sandboxed virtualization) and during execution (monitoring the operations performed) for behaviours commonly exhibited by unwanted software[8].

Because system operation is monitored at quite a low level, known attacks can often be prevented by signature based methods if the signature is in the database, however heuristic methods are not as accurate. The top antivirus software at present has detection rates in the 95-99% range for known viruses, but this rate is considerably lower in some cases[11].

The different types of software (antivirus, antimalware and antispyware) use similar mechanisms of operation, but target slightly different sets of unwanted software. Antispyware and antimalware software monitor for software (or data files) that may store or disclose personal information or log user activity, or allow another to monitor and control the system (including software deeply hidden in the system (rootkits)). Antivirus systems target more general potentially dangerous software, such as trojans, worms, and viruses. Although antivirus software

[8] An example of this could include software trying to write its own code into system executables.

sometimes monitors for the same type of software as antimalware/antispyware software, it does not always do so. Some types of software such as adware[9] or corporate remote monitoring software are not always unwanted or dangerous and so they are not always removed.

Web Security Software and Software Firewalls have similarities to antivirus software, but are more targeted in their threat protection model. Where antivirus software monitors system Input/Output, web security software and software firewalls manage software network access and the content that flows through the network. This is undertaken by intercepting traffic to and from the network and performing various checks upon it. These checks include checking for access to blacklisted sites, performing antivirus scans on incoming traffic, and checking for outgoing personal information in some cases.

2.6.3 Centralized or gateway based mitigation approaches

Most networks have a constrained number of perimeter entry and exit points that all incoming data must pass through. Monitoring these perimeter locations promises to prevent external threats from entry and internal threats from exfiltrating data. The monitoring may occur either at the network layers, as in routers and firewalls, or at the application level in the case of web gateways and application layer firewalls.

Routers and Firewalls can monitor the traffic flowing through them for patterns of malicious traffic, and block it if necessary. However, they cannot inspect application level content such as state in a meaningful way without adding the functionality of web gateways and application layer firewalls.

Application layer firewalls

monitor network traffic with an understanding of the workings of the higher level protocols and content contained within it. A *secure web gateway* is a special case of an application layer firewall that is designed to protect against threats from web traffic. A proxy with filtering rules is a simplified secure web gateway, however a comprehensive secure web gateway attempts to mitigate many different types of web attack by monitoring web traffic and intercepting the traffic as either as a network firewall or as a transparent web proxy. The common secure web gateway's approach to the detection of undesirable content is simple blocking via a blacklist approach.

[9] Software that is installed as a condition of running some other software and displays ads to the user. Many unwanted browser toolbars follow this model

The inspection of encrypted content presents a possible problem to secure web gateways. If a secure connection is terminated, decrypted, inspected, and re-encrypted then client errors occur (by design). The gateway may reencrypt without error if it signs transactions with a certificate signed by a mutually trusted root CA[10].

2.6.4 Other Protection Models

Other protection models in literature include cloud based and immune response models. Cloud based models perform content scanning operations on an external server and then act upon the reported results. They are useful for system with limited processing power and for situations where a single point of monitoring is useful. For more information see: [118, 192, 117].

Immune and Intelligent Agent Models function through the use of applying biologically inspired immune models to computer security. They utilize numerous mechanisms including the use of autonomous agents to provide protection in a decentralized fashion. For more information see [85, 65, 49]

2.7 Related Research

2.7.1 Threat Prevalance Surveys

Many different research labs and vendors of both software and hardware regularly release various *security reports*, *threat surveys*, or *annual reports* on web threats. Varying greatly in quality, these must be approached with caution as they can be heavily biased. Nevertheless, many are released by well-known and well-respected organisations. A careful overview that considers multiple such works can provide a useful overview of the state of web security, as the authoring groups are in positions that enable them to collectively monitor the vast majority of malicious activity detected worldwide.

The organizations that release these reports consist of groups within large software companies, security product vendors, and governmental or industry bodies. The motivations and background for each source should be considered when assessing any information given. While reports from general vendors, government, and other industry organizations are likely to be reasonably correct, those from security vendors must be approached with caution (after all, they need to sell you the problem they claim to solve). There is insufficient space for discussion of

[10]This can be undertaken either by installing either an additional root certificate on the client or a certificate on the proxy signed by an already trusted root.

every report consulted for background information used in this work, but a general overview of the most important and well regarded sources referred to in this work is given in Appendix D. Where information from a specific report is used, that report is cited individually.

2.7.2 Fraudulent content and countermeasures

Fraudulent content in the form of fake storefronts or phishing sites is widespread, largely targeting financial institutions [103], and is hosted particularly on compromised legitimate web sites [73] [11]. Fraudulent website protection techniques in the literature come in two forms: reputation based (blacklists and web of trust approaches), and automated score based approaches.

Blacklisting and whitelisting approaches aim to prevent users falling victim to fraud by preventing (or warning about) access to bad websites and only allowing access to good sites respectively. Blacklists are very effective at preventing attacks by sites on their bad list, but necessarily only as effective as their list is. Blacklist update speed is crucial to the effectiveness of such an approach[155]. Blacklisting is widely discussed [155, 95] and implemented [193]. Blacklisting is used in many current implementations of web software such as Google Safe Browsing (used by Firefox), Microsoft Internet Explorer's Phishing Filter, and many web security toolbars such as McAfee Siteadvisor[158].

Web of trust Web of trust approaches break conceptually from the centralized trust model by using input from multiple users to make decisions – most still work in a blacklist like approach, though by collating responses centrally. The My Web Of Trust (MyWOT)[114] web browser addon uses this principle.

Automated Fraudulent content detection functions in a similar way to blacklisting, but extends the trust model beyond a simple binary decision. While blacklisting is inherently reputation based, it is non-granular and can only return a true or false result. A score can be generated either by manual or automated means, and in return the score can be either processed automatically or by the user. Automatic score generation can may be undertaken by similarity analysis of page or URL features [187, 81, 95] or by examining other page features such as the use of HTTPS (most phishing sites do not use HTTPS, yet legitimate sites should use it to request login information).

[11] Refer to the sources in appendix D for specific numbers, as by the time this work is read the information used will be outdated.

2.7.3 Malicious Web Sites

Much of this work is concerned with protection against technical threats, thus the following from the literature is important: malicious website prevalence and attack mechanisms, methods of malicious web site detection, and methods of protection from the effects of malicious web sites.

Malicious Web Site Prevalance

Malicious website studies have been undertaken by many groups over the past half decade or so, and the publicly available statistics and studies are compiled from two main study methodologies: removal tool statistics and crawler based studies.

Removal and protection tool statistics consist of statistics released by software vendors regarding the number of attacks detected and blocked. These can be over quite a large sample set and userbase, but can only include information about attacks which are within the scope of the associated tools' ability to detect.

Microsoft produces biannual *Security Intelligence Reports* (SIR), which contain compiled information about malware removed from Windows machines with the Malicious Software Removal Tool (MSRT) or blocked by the Internet Explorer Smartscreen website filter. The MSRT runs automatically on millions of Windows PCs worldwide to remove a limited (by no means comprehensive) number of malicious software families. Each SIR gives the information broken down by many different variables, including Windows versions, malware found, and geographic location. For instance, in the first half of 2010 the SIR volume 9[103] reports that websites with domains in the .cn, .ru, .de and .uk top-level domains contained approximately 13% of all malware sites hosted with infection rates of .67%, .23%, .1% and .1% respectively.

Symantec reports that overall in 2009 approximately 5% of websites were classed as risky (this is a less specific criterion than the Microsoft statistics) [83]. The potential bias in each source should be considered, as it could be argued that Microsoft has a business interest in downplaying malware rates while Symantec has an interest in the opposite.

Crawler Based Studies are studies undertaken where an automated system crawls websites (either working through a list, or collecting links off websites and visiting those) and checks for infected files or infections acquired without intervention (via drive-by-downloads). Infected

binary file rates were assessed in [108], who report that after automatically collecting binary files from a large number of websites approximately 4.4% were flagged as Spyware infected.

Global crawler studies report drive by download rates rates around three tenths of one percent but differ slightly in approach and security settings. [108] report approximately 0.2% of websites resulted in an infection via a drive-by-download while [128] report rate of approximately 0.3% after a thorough analysis to remove false positives. [123] reported a drive by download rate of around 0.5% which although higher, is in the same region.

Search engine results are often targeted by attackers as a vector to use to get users to their sites. [103] reports that 0.2% of Bing results shown to users contain malicious content, while [128] reports that 1.3% of Google results pages contain malicious links.

For a New Zealand based perspective, a study by Seifert et al[148] in 2008 recorded a malicious web server rate of 0.2% in the .nz domain space, and this is consistent with other reports that .nz domains have a low infection rate (e.g. [103]).

Malicious Web Site Attack Methods

Malicious web sites successfully attack by somehow exposing vulnerable users to attack code that then executes an attack to infect their system. We briefly discuss these steps as user exposure, exploitable user vulnerability, attack execution, and infection. Although we will not discuss them here these steps can be automated by "exploit packs" installed on web servers as discussed in [5, 80, 93].

User Exposure User exposure occurs via three main methods: *Blackhat Search Engine Optimization* (Blackhat SEO), Malicious Advertising (*malvertizing*), and the compromise of a legitimate site. Blackhat SEO is the process of compromising search engine results in a way that users searching for popular things get links to sites operated by attackers (discussed in [5, 64, 73, 97, 195]). Malvertizing is the practise of somehow getting advertisements controlled by the attacker on websites and can occur either directly (as discussed in the introduction), or via the compromise of multi-site advertising networks (discussed in [128, 129, 101, 64, 143]). The compromise of a legitimate site can occur via some form of credential compromise such as hosting details, a website weakness such as SQL injection, or the compromise of a web host (discussed in [39][166][171]).

Exploitable Vulnerability Once the user has been exposed to attack code, the next step is to somehow execute the attack. This requires some form of vulnerability which can be either human (user), or technical. The user may be exploited if they can somehow be convinced into running the attacking executable and potentially disabling other technical defenses.

Technical vulnerabilites are the cause of the automated attacks (known as drive by downloads) and are usually in the form of unpatched remote code execution bugs (such as buffer overflows) in software. Although vulnerabilities that are publicly known but for which no patch exists (so called zero-day vulnerabilities) are a risk at times and difficult to protect against, the particularly vulnerable users are those who remain unpatched after a patch is available. According to Frei et. al. [56] 45% of browsers in use at the time were not the latest release available, while from [103] it can be inferred that this is improving (<25% of windows users do not have automatic update enabled in mid-2010 and although this would not protect them against all threats, it would certainly help).

Attack Execution usually occurs when an executable written by the attacker is run. This executable may be contained in the original exploit code, it may be contained on another website to which the browser is redirected, or it may be in the form of a second executable which is downloaded by a *dropper* that the exploit first downloaded.

Embedded executables are the simplest case conceptually, however they are not often used as they are the least flexible for campaigns that infect a wide variety of websites. Redirection is the most commonly used attack method [128, 147, 37], as it allows the attacker to modify the attack binary without needing to change it on every infected site. Droppers are similar in approach to redirection, but rather than the redirection being handled by the web browser, it is handled by an executable that is first downloaded (the claim could be made the the running of the dropper is the culmination of the attack's success).

Malicious Web Site Detection

The detection and study of malicious web sites in the literature has primarily been undertaken via crawler based studies. The crawlers used are of three forms: virus scanners, static heuristics, and honeypots.

Virus and spyware based crawlers as discussed in Moshchuk et. al.[108] reportedly exhibit a poor detection rate[129] (high false negative rate), and were not used in this work as they also have a smaller research footprint. Static heuristics, as discussed in [152], look for page

characteristics that are commonly found on malicious pages. They achieve higher performance levels than honeypot based methods, but also report high false positive rates.

Crawlers that use client honeypot based methods (as used in this work) are very common in the literature but these are considerations that must be taken into account regarding their use. A client honeypot is a system that emulates a client system in some capacity to detect attacks, and they come in two main forms: high-interaction and low-interaction [126, 115]. A low-interaction honeypot (e.g. Phoneyc [115]) emulates only a subset of the target system, and exhibits a high level of performance, but can have problems detecting unknown attacks [84]. High-interaction honeypots are comprised of systems running a full software environment and while they can exhibit a good degree of target emulation, they can have slow performance when compared to other methods due to the additional resource overheads[151].

High-interaction based honeypots[12], were used in the bulk of the crawler based studies in the literature and in this project. The main studies and the name of the honeypot software used are[13]:

- Shelia[16]
- Strider Honeymonkey[182]
- Honeyclient (Also known as MITRE Honeyclient)
- Capture-HPC[72, 146, 67]
- <Unnamed system at Google > Provos[129]

This project makes extensive use of Capture-HPC which is discussed in [146, 67, 148, 151], and used in [84, 37, 45, 147]

2.7.4 Related Protection Mechanisms in the Literature

Protection of clients against malicious websites consists is usually undertaken with the use of software updating, antivirus, and web security suites. Research into infection detection methods before and after infection is ongoing [129].

This work is concerned with protections that do not reside on client systems, particularly those that reside on proxy servers. Among the particular tools in the literature that present aspects of interest are BrowserShiled, SpyProxy, and Blade. Browsershield as discussed in Reis et. al. [135] rewrites page content to enforce "*vulnerability driven filtering*. That is, it removes

[12]While honeypots may be run on a physical system the use of Virtual Machines (as discussed in 2.4.3) have allowed the use of high interaction virtual honeypots[182]

[13]Other relevant literature includes [128, 151, 148, 108, 130, 84, 115, 106, 82, 83]

code which appears to be attempting to use known exploits to attack clients so that the attack cannot succeed. Spyproxy (discussed in Moshchuk et. al. [107]) first sends content to client honeypots and checks for malicious activity of websites before sending the content to users. Block All Drive-by download Exploits (BLADE), as discussed by Lu et. al. [91][92], uses software installed upon the system to perform what the authors term *unconsented-content execution prevention* (put simply it attempts to block executable code unless explicit user permission is given).

2.7.5 Related System - Comcast Constant Guard

The American Internet Service Provider Comcast has recently deployed a system which many of the early ideas in this work were derived from. Discussed in IETF informationals, there are ten versions available on the IETF website, eight denoted drafts [20, 22, 23, 24, 25, 26, 27, 28, 29] and one that is not denoted a draft [21] [14]. Comcast deployed the system under the product name *Constant Guard* in October 2010 [33].

Constant Guard was designed to enable Comcast to notify customers of Bot infection, as discussed in the IETF Informational *Recommendations for the Remediation of Bots in ISP Networks* (latest version:[90]). Constant Guard uses a similar architecture and similar software to that we use for the system in this work, however it only undertakes inline notification, and does not in any way attempt to protect users actively. The warning is supplemented with an email to the ISP held email account, and a screenshot of the Constant Guard user alert is shown in fig 2.15:

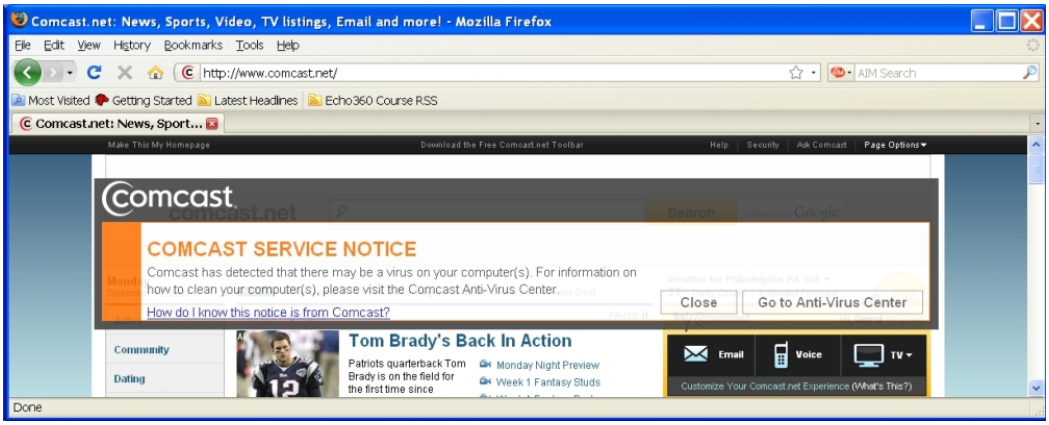

Figure 2.15: The inline warning web page overlay as used in Comcast Comstant Guard from [121]

[14] This project began around the time of the first draft.

Chapter 3

System Development

The web browser is a common point of network attack, as content can both enter and leave a network through the web browser. Users may be completely unable to detect anything is amiss, as often their computer's defensive measures are silently disabled. It is one thing to find information regarding infection or danger, and quite another to make use of it.

The most common response to security alerts is to log them and alert system administrators, but this is inherently retrospective and can come too late to make a change. Among other possibilities are automated responses to either alert users directly or to respond to the threats. Those options are investigated in this project to develop a secure web gateway to prevent clients from external threats. Protection against exfiltration of data from internal threats has been relegated to future work (section 7.2.3).

The Secure Web Gateway developed for this project deploys automated responses to both alert users and deal with some threats directly. One way to do this would be to set up proxy servers that users can choose to use on an opt-in basis. These specific proxy servers contain these security controls, thus not affecting in any way the browsing experience of users who choose not to use them; however, this is a policy decision, not a technical one, and is beyond the scope of this project.

3.1 Goals

Current approaches to securing web access from external threats take two forms: client side solutions, and centralized proprietary systems (as discussed in section 2.6). Our goal in this project was to develop and test a centralized system (a so-called secure web gateway) that is open source and standards based. This will perform operations within the following general

classes (which we will briefly discuss next):

- User Messaging
- User Notifications and Security Warnings (Inline)
- Automatic Mitigation of Threats

3.1.1 User Messaging

User awareness is an important part of security. But care must be taken to ensure security warnings are not so obtrusive that they either get perceived as noise and ignored, or annoy users into disabling the warnings. User notification can be achieved through different methods, and different workflows, each with different considerations.

A notification method is needed that can reach users directly, noticeably, and immediately. Email serves this purpose in many businesses, but it is still not immediate and can be easily lost to a spam filter. The type of Internet use that most users will undertake most regularly outside of email is website browsing, however it is inherently demand-driven – users seek out specific content and cannot be assumed to be checking a specific information source regularly. One solution to this problem is that used by Comcast in Constant Guard (as discussed in section 2.7.5), which uses the Internet Content Adaptation Protocol (ICAP) to insert messages in web traffic. The ICAP protocol enables web traffic to be modified on the fly as it passes through a gateway system, thus messages to users can be inserted on any website they visit.

The notification can be carried via a side channel (such as email), it can be displayed in a separate area to the item to which it is relevant (such as a pop-up dialog), or it can be presented inline. Side channel notifications such as email and telephone are used by many ISPs to contact customers, but can be slow to reach the customer (as not every customer compulsively checks their ISP email account). Pop-ups and out of workflow alerts can require additional software, and sometimes be missed as the user is not concentrating on them. Inline notifications have some trust issues, as it can be impossible to verify the source of the notification, thus leaving users open to abuse such as phishing style attacks.

We developed inline notification due to the nature of the technologies we are using, however, multiple types of notification should be used in a full-fledged notification system. The following user targeting styles were implemented for the user messaging functionality:

- Message individual users

- Message the entire userbase
- Message the entire userbase based upon a database rule of an arbitrary form (E.g. all users in Wellington)

User notification may also be desirable in some situations that are not necessarily security related, for instance, as a method to effectively reach large numbers of users who are not easy to contact via direct mediums such as email or a website, or broadcast mediums such as television or radio. Broadly targeted inline-modification could be a useful method for system wide outage alerts, or for more general situations such as natural disaster notification. Narrowly targeted inline notification could be useful to service providers in scenarios such as account compromise, data cap issues, or billing problems.

3.1.2 User Notifications and Security Warnings (inline)

Messaging users has many security uses, but it is inherently based in the past, as it requires the message to be added to the database before it is shown. Security warnings of a more immediate nature are also used in some widely deployed products. For instance, the Firefox web browser uses an inline notification (see fig 3.1 below) for dangerous websites which shows many features that are relevant to our user notification implementation.

Figure 3.1: The Firefox ”reported attack site” primary warning

This particular warning is very difficult to naively ignore – in order to bypass it the user must acknowledge the message and consciously find the small ”ignore this warning” link in the bottom right. Furthermore, should the user click ”ignore this warning”, a further warning

overlay with options to ”Get me out of here” or report a false positive is displayed at the top of the window as shown in fig 3.2.

Figure 3.2: The Firefox ”reported attack site” warning bar, which remains visible if the user continues past the warning

The use of content modification, allowing quick communication with users and warnings about detection of malware on customers' systems, is already demonstrated in use by the American Internet Service Provider Comcast in their *Constant Guard* system (discussed in section 2.7.5).

Among the aims of this project are the deployment of inline security warnings to users to alert them to the following problems:

- Browser Version Outdated
- Plugin Version Outdated (where possible with simple checks)

3.1.3 Automatic Threat Mitigation

The Comcast Denver (Constant Guard, or CG) system offers notification to users, but it does not offer any protections to them. We built upon their ideas in a similar system to offer users some security protection measures in addition to notification. Specifically, our system attempts to automatically detect and notify the user of malicious content, and to perform selective alteration of content to sanitize it. This will violate some of the documented core requirements of the Comcast CG system, but the system we are developing is intended for deployment in an environment with a higher level of control (such as a business or corporate environment). [1]

Users can and should be notified of any content alteration, and this can occur in various ways, ranging from usage agreements, to small messages, to page overlays or complete page replacement. The ICAP protocol can also be used for modification of active page content such as JavaScript because it is content agnostic, and can inspect all unencrypted content. This can

[1] We are concerned with the technical issues, not the potential legal, copyright or ethical issues which may arise from such a deployment. We leave those issues to any who choose to deploy such a system. Furthermore, many of these issues are already prevalent with technologies such as caching web proxies.

also be used to block bandwidth intensive page content (such as many types of advertisement), cutting bandwidth usage, and mitigating some types of cross domain attacks.

Our general architecture offers proof of concept protection against some classes of threats. The specific threats we intend to investigate protection against are:

- Objectionable (keyword based) textual content
- Browser exploitation and drive by downloads
- Dangerous filetypes
- Virus downloads
- Phishing and site impersonation
- Cross site scripting

3.2 Hypotheses and Assumptions

The hypotheses evaluated in this project are inherently simple, as this work is concerned with the development and evaluations of a system using existing technologies.

3.2.1 Base Assumptions

There are a few assumptions that are required to establish a context for the remainder of this work. These primarily include, but are not limited to those below. For further detail on each see the appropriate referenced material:

- User notifications to end users can be used to inform their decisions and thus improve their security choices [149, 87, 88].
- Some classes of web threats may be mitigated to a degree through automated means[167, 45, 37, 44, 180].
- Open source tools allow for a good degree of software transparency and flexibility[89, 70, 189].

3.2.2 Primary Hypotheses

- (The open source tools used for and with) The Internet Content Adaptation Protocol can be used to modify, sanitise, or deactivate specific web content, and thus act as an application layer Intrusion Protection System (IPS) for World Wide Web content.

This work did not use null hypotheses. As a project focused on the development of a technology rather than understanding the working of a technology, it was felt that a null hypothesis such as "The ICAP protocol cannot be used to act as an application layer IPS for WWW content" would be both justifiably false (since ICAP can block all web, which is traffic the most trivial IPS operation) and would add nothing to the pool of knowledge.

3.2.3 Secondary Hypotheses

- The modification, sanitization, or deactivation of web content as performed by ICAP is able to improve web browsing security by reducing exposure to threats.

- The modification, sanitization, or deactivation of web content as performed by ICAP may be performed without a large or unacceptable impact upon system throughput, as request delays may negatively affect user experience.

3.3 Novel Aspects

Our work has novel aspects above and beyond existing secure web gateway products, some of which are shared with the Comcast system, and others which are unique to this project. Specifically these are:

- Open Source and Standards Based
- Integration Simplicity
  - E.g. Security monitoring systems put messages in the database for infected users.
- (Pseudo) Stateful
- Modular Security
- Novel Effectiveness Testing Methodology

Open Source and Standards Based - the system developed uses entirely open source tools, and the operation of it fits documented expectations for the protocols used. This enables inspection, expansion, and modification of the setup and operation by any user thus enabling better security assurance. A similar product, Untangle [175], was found late into the project and shares some similarities.

Integration Simplicity - an implication of the open source and standards based aspect of our system, because it fits standards and the operation can be inspected, the integration of new functionality, systems, or clients should be simplified.

Modular Security - a result of the way the ICAP server used (Greasyspoon) functions. For security this results in each security operation being able to be treated independently, and for operations to be modified, enabled, and disabled on the fly.

(Pseudo) Stateful - HTTP, ICAP, and the related protocols are inherently stateless [2], and each request/response is treated independently. Many security threats (such as XSS) rely upon interactions over multiple requests, and thus cannot be adequately prevented with stateless security measures. By coupling request and response security modules pseudo stateful security operations can be performed via the ICAP server.

3.4 Requirements

The secure web gateway developed protects HTTP traffic only, and does not integrate with a firewall. The Comcast system is architecturally similar with a history spanning many versions[3], and has some overlapping usage criteria, so some requirements also overlap. Our requirements are not identical and we do in fact need to deliberately violate some of their requirements to make our system functional. We will therefore assess which of their requirements are applicable to this project, and add our own.

3.4.1 Requirements as relevant from [21]

The Comcast System Requirements are given in full including details in appendix C. Those which are relevant to our system are discussed below:

[2] Some measures like cookies attempt to work around this and add the concept of sessions.

[3] There are 9 versions on the IETF website[20, 22, 23, 24, 25, 26, 27, 28, 29, 21], of which 8 were published while this work was in progress)

```
REQ1:   TCP Port 80
REQ2:   Block Listing [Blacklist some sites from modification]
REQ3:   Instant Messaging (IM)
REQ4:   Handling of Active Sessions
REQ5:   No TCP Resets
REQ6:   Non-Disruptive
REQ7:   Notification Acknowledgement
REQ9:   Unexpected Content
REQ10:  No Caching
```

System Requirements 1: - General [Abridged – Full Details in Appendix C]

Our system should follow requirements 1, 2, 3, 4, 5, 6, 7, 9, and 10, however it is necessary that it not follow requirements 8 and 11 due to the extra functionality we intend to include that necessarily violates those rules.

```
REQ12:  Open-Source Software
REQ13:  ICAP Client
REQ14:  Access Control
```

System Requirements 2: - Proxy [Abridged – Full Details in Appendix C]

Our system will follow these requirements (12-14), as we are using the same web proxy (Squid). There is no need to test these requirements.

```
REQ15:  Request and Response Support
REQ17:  Multiple Notification [or Modification] Types
REQ18:  Simultaneous Differing Notifications [or Modifications]
```

System Requirements 3: - ICAP Server [Abridged – Full Details in Appendix C]

Requirement 15 will be followed as we will use the same software, we should meet requirements 17 and 18 which we have broadened to cover general modification and not simply message insertion. Requirement 16 is inapplicable to us, as our notifications change with the web page input which we do not control.

```
REQ19:  Messaging Service
REQ20:  Process Acknowledgments
REQ21:  Ensure Notification Targeting Accuracy
REQ22:  Keep Records for Customer Support
```

System Requirements 4: - ICAP Server [Abridged – Full Details in Appendix C]

We intend to follow requirements 19 - 22.

3.4.2 Additional Requirements

In addition to the requirements above out system will require a few more:

- **REQ A1: User Override**
  - Users must have the ability to safely bypass any warnings and perform the desired action in a potentially unsafe manner

- **REQ A2: Automated Update**
  - The system must be capable of unattended signature updating for antivirus

- **REQ A3: Acceptable Delay**
  -The system must NOT cause unacceptable delays for users (More specific than REQ6)

- **REQ A4: Throughput and Concurrency**
  - The system must support a reasonable throughput and number of concurrent connections before performance degradation

- **REQ A5: Improved Security against malware attacks**
  - The system must provide measurably decreased infection rates for clients that use it as a web proxy.

3.5 System Architecture

Because this project is primarily concerned with extending the functionality offered in the Comcast Web Notification System, we based the core components around the components used in that implementation. Thus, we developed our system upon the core components of a Squid Web proxy and a Greasyspoon ICAP server. We chose to use ICAP directly for our notifications, and to not use an external webserver for this, as Chung et al. [21] did with a Tomcat server. This core was integrated with the following other software tools to allow additional functionality required: a MySQL database server, a Clam Antivirus server, and a Freeradius authentication server. Additionally, IPTables was used on the network gateway to provide Network Address Translation (NAT) access to the Internet. An overview of the system architecture is shown in fig 3.3 below.

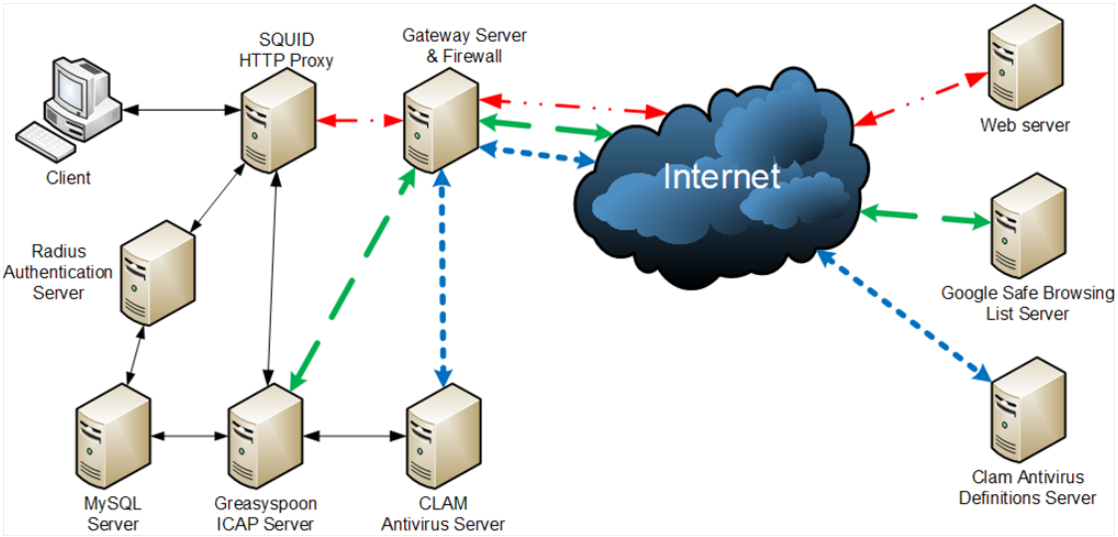

Figure 3.3: A logical architectural overview of System developed

Table 3.1: Testbed architecture components & network details

Machine (OS)[a]	**Component**	**(Interface) Address : Port**
1 (Ubuntu)	Firewall / Gateway (External Side)	(Eth0) 132.181.19.2
	Firewall / Gateway (Internal Side)	(Eth1) 10.1.1.1
	Clam Antivirus	10.1.1.1
2 (Ubuntu)	Squid HTTP Proxy	10.1.1.2 3128
	SafeSquid HTTP Proxy	10.1.1.2 8080
3 (XP)	Greasyspoon ICAP Server	10.1.1.3 1344
	GreasySpoon Admin	N/A (client)
4 (Ubuntu)	MySQL Server	10.1.1.5 : 3306
(Ubuntu)	Free Radius Server	10.1.1.20 : 1812

[a] Ubuntu 10.04, Windows XP professional Service Pack 3

3.5.1 Component Overview

Similar to the Comcast System, but with a few additions.

- **Squid Proxy Server** - The HTTP gateway that handles all HTTP traffic.
- **Greasyspoon ICAP Server** - Interfaces with the HTTP Proxy and gives content modification Instructions to the Proxy. Also communicates with remote third party servers for blacklist checking.

- **MySQL Database Server** - Stores user, message, and configuration information and interfaces with the ICAP and authentication servers.
- **Antivirus Server** - Interfaces with the ICAP server to perform antivirus scanning. Also interfaces with a remote third party server to update signatures.
- **Radius Authentication Server** - Interfaces with Squid proxy to check with the database server on user authentication and credentials.

3.5.2 Additional Software Used

The following software was also used:

- **IPtables Firewall** - To allow SQUID proxy transparency in later testing.
- **RADIUS Pluggable Authentication Module (PAM) for SQUID** - Allowed Squid to authenticate users using RADIUS.
- **Google Safe browsing API with jGoogleSafeBrowsing [154]** - Used to check URLs against Google's bad site blacklist.
- **Ubuntu Linux 9.10/10.04** - Base Operating System for ClamAV, MySQL, and network gateway.
- **Windows XP** - Base OS for Greasyspoon server.

3.6 Main Component Details

3.6.1 Squid

Squid is an open source (GNU General Public License (GPL)) web caching proxy server that is derived from a codebase which has been in development and use for around 15 years. The initial project was the Harvest cache, and Squid was one of the projects that forked off the Harvest Project. The initial Squid project itself was funded under grant funding, but after grant funding ran out development was continued through volunteer donations and "occasional commercial interest" [163].

The latest major version of Squid (version 3.x) offers support for ICAP, which is a requirement for this project. Squid also supports transparent mode, where web clients need not be configured to use a proxy. While it may be desirable for clients to opt in to using the secure

proxy due to privacy concerns, if circumvention is not desirable (i.e. a corporate environment) then transparency support is a requirement. Most of the additional Squid features are irrelevant as we can also perform them through the use of the Internet Content Adaptation Protocol (ICAP).

3.6.2 Greasyspoon

Greasyspoon is the ICAP server used in this projects, it "*allows you to manipulate HTTP traffic by creating simple scripts in various possible languages*" [105]. GreasySpoon is a cross-platform Java application and supports scripts written in Java and JavaScript/ECMAscript initially. Support for other scripting languages (such as Ruby or Python) can be added by using Java JSR223 (Java Specification Requests) modules to interoperate other interpreters with the Java interpreter. The greasyspoon logical architecture is shown in figure 3.6.2

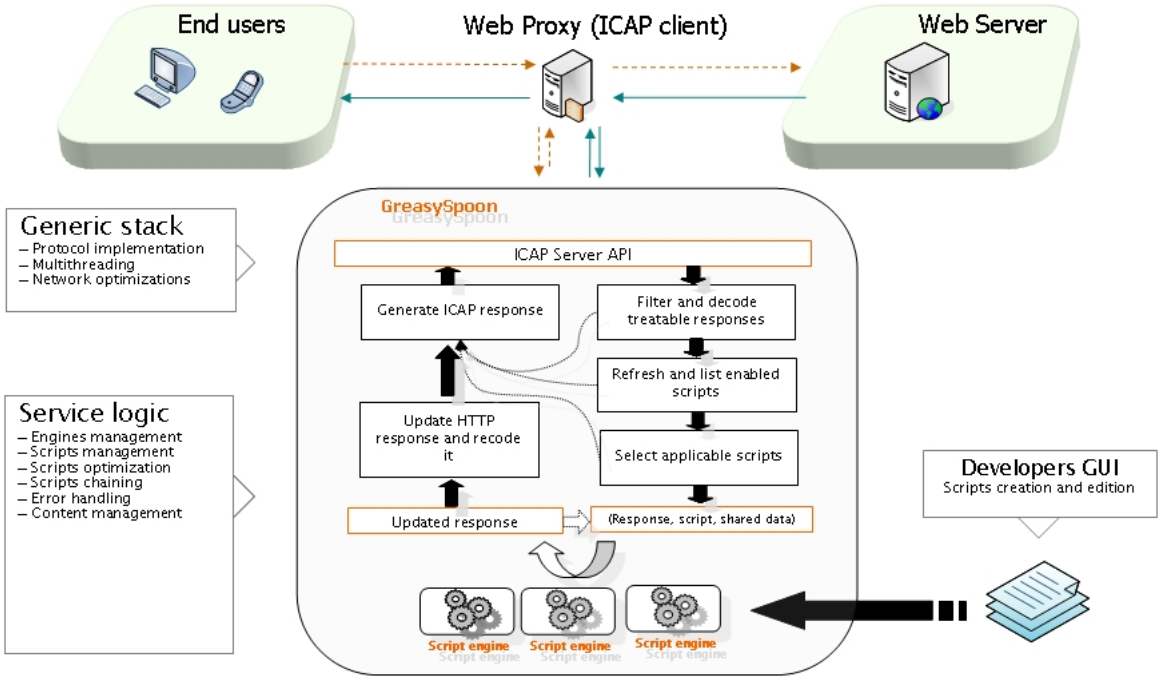

Figure 3.4: The Greasyspoon architecture – from the website [105]

Greasyspoon is controlled via a web based interface, and is designed to be easy to use and administer. The web interface provides graphical reports, access to system settings, and management of scripts. Greasyspoon scripts come in two types to correspond with the two ICAP modification methods: response modification scripts, and request modification scripts. Because they are interpreted, scripts can be enabled, disabled, and modified on the fly, and Greasyspoon provides an in-browser script editor that includes syntax highlighting and auto complete. This

wqorkflow for Greasyspoon is illustrated in figure 3.6.2:

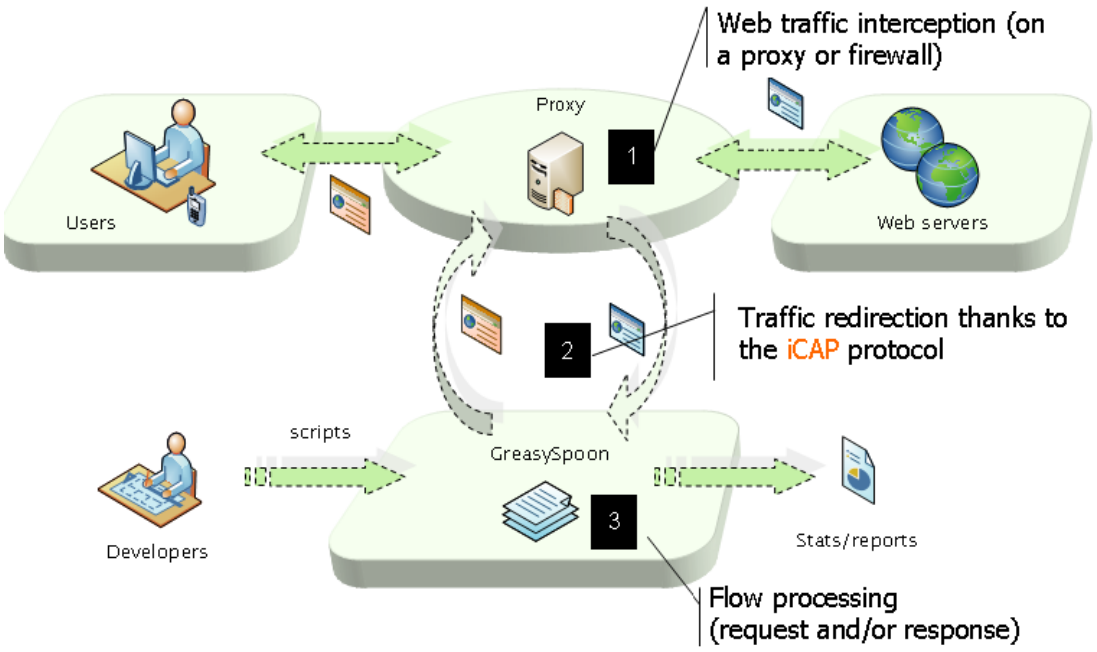

Figure 3.5: The Greasyspoon ICAP server basic architecture and operational workflow [105]

Scripts contain headers that define some basic parameters surrounding their use that are defined in a standard format. Headers are defined inside a comment (appropriate to the scripting language) and consist of <parameter name> <parameter value>. The relevant control parameters include *status* (on or off), *include* and *exclude* (which define the URLS to which the script is or is not applied, defined by regular expressions), and *timeout* (defines a script specific operation timeout if the default is not desirable).

3.6.3 MySQL

MySQL is a widely used Open Source Relational Database Management System (RDBMS). MySQL is used so pervasively it is not felt necessary to go into detail here: for more detail see the official MySQL website [113]

3.6.4 ClamAV

ClamAV is a command line virus scanner for Unix like systems designed for use on Internet mail servers It was acquired by Sourcefire in 2007, but continues to be developed under an open source license [30].

...ClamAV is one of the most commonly-used open source antivirus and antimalware prod-

ucts in the world. ... ClamAV ... is currently integrated within leading enterprise solutions, including Unified Threat Management Systems (UTM), Secure Web Gateways and Secure Mail Gateways, to identify deeply embedded threats such as viruses, trojans, spyware, and other forms of malware.[161]

ClamAV also allows remote file scaning via TCP sockets, which is very useful for this project. Unfortunately ClamAV has poor detection rates [10], but as probably the only open source antivirus product available (we are unaware of any other), the choice was constrained.

3.7 Component Configurations

Component configurations were kept as close to the default as possible, with modifications made to interoperation of the system components.

3.7.1 Squid

Squid was set up in with the following changes from a default configuration (initially non-transparent):

- Caching completely disabled.
- All ports above 1024 added to the safe_ports list.
- ICAP Settings:
  - ICAP enabled.
  - ICAP bypass on error disabled.
  - Squid Sends ICAP server the Client IP and Username
- Radius PAM for SQUID set to use the FreeRadius Server.

The appropriate squid.conf file is given in Appendix E.1.1.

3.7.2 Greasyspoon

Required Modifications to Greasyspoon Greasyspoon source required some modifications for functionality we desired. Namely:

- Get content of requests/responses in binary
- Accept all file types (previously ignored non-textual types)

The configuration files used for Greasyspoon are given in Appendix E.1.2 (for greasyspoon.ini) and Appendix E.1.2 (for icapserver.conf).

3.7.3 MySQL

MySQL permissions were set open to all for simplicity – not a real world implementation but irrelevant to the effectiveness of the project. For Freeradius, the default schemas were used, and the schema for user messaging is found in Appendix G.

3.7.4 ClamAV

Signatures were automatically and remotely updated every 2 hours via running the Freshclam service. The configuration of ClamAV in clamd.conf was mostly the default, with the following notable settings:

- TCPSocket: 3310
- TCPAddr: 10.1.1.2
- MaxThreads: 25 – Scan 25 files concurrently, queue others
- ReadTimeout: 120 – Timeout if no data ion 2 minutes
- StreamMaxLength: 20M – File Larger than 20MB skipped
- ScanOLE2: enabled – Scan office macros
- ScanHTML: enabled – scan HTML files
- ScanMail: enabled - scan mail files
- MailFollowURLs: disabled – do not follow URLs in files (too intensive)
- ScanArchive: enabled – scan archives

3.7.5 Freeradius

Freeradius used the MySQL server's database. There were no other relevant configuration changes.

3.7.6 Host Operating Systems

Linux (Ubuntu 9.10/10.04) or Windows XP 32bit Service Pack 3

Base Operating Systems were used for various components, no relevant reconfiguration was necessary.

3.8 General Script Operations

The scripts developed perform a variety of request/response inspection and modification operations, however, these are all composed of one or more of the following basic operations. These same cases are also used in performance testing as representative examples.

1. **Request Header insertion** - Unconditionally insert a header field into the HTTP request sent to the web server
2. **Request Header Modification** - Inspect and, if necessary, alter/remove/add some combination of HTTP headers in the request
3. **Request URL Conditional Change** - Inspect and, if necessary, alter the request URL of the request sent.
4. **Request to Response Modification** - Inspect the request and, if necessary, send the client a response directly without the request being serviced externally.
5. **Response Content Modification (Static Conditional)** - Inspect and, if necessary, alter some content in the HTTP response body
6. **Response Content Modification (Caching DB lookup)** - Inspect and, if necessary, alter some content in the HTTP response body as controlled by a database server. The database is not checked on every call, but the change is cached and updated periodically
7. **Response Content Modification via Remote Server** - Response body is sent to a remote server which instructs the ICAP server in how to proceed.

3.8.1 Feature Set

The specific features implemented (discussed later) correspond to basic features that can be found in enterprise secure web gateways, client web security packages, and the Comcast system, as well as some unique features. How they fit in the broader context and overlap is illustrated in figure 3.6.

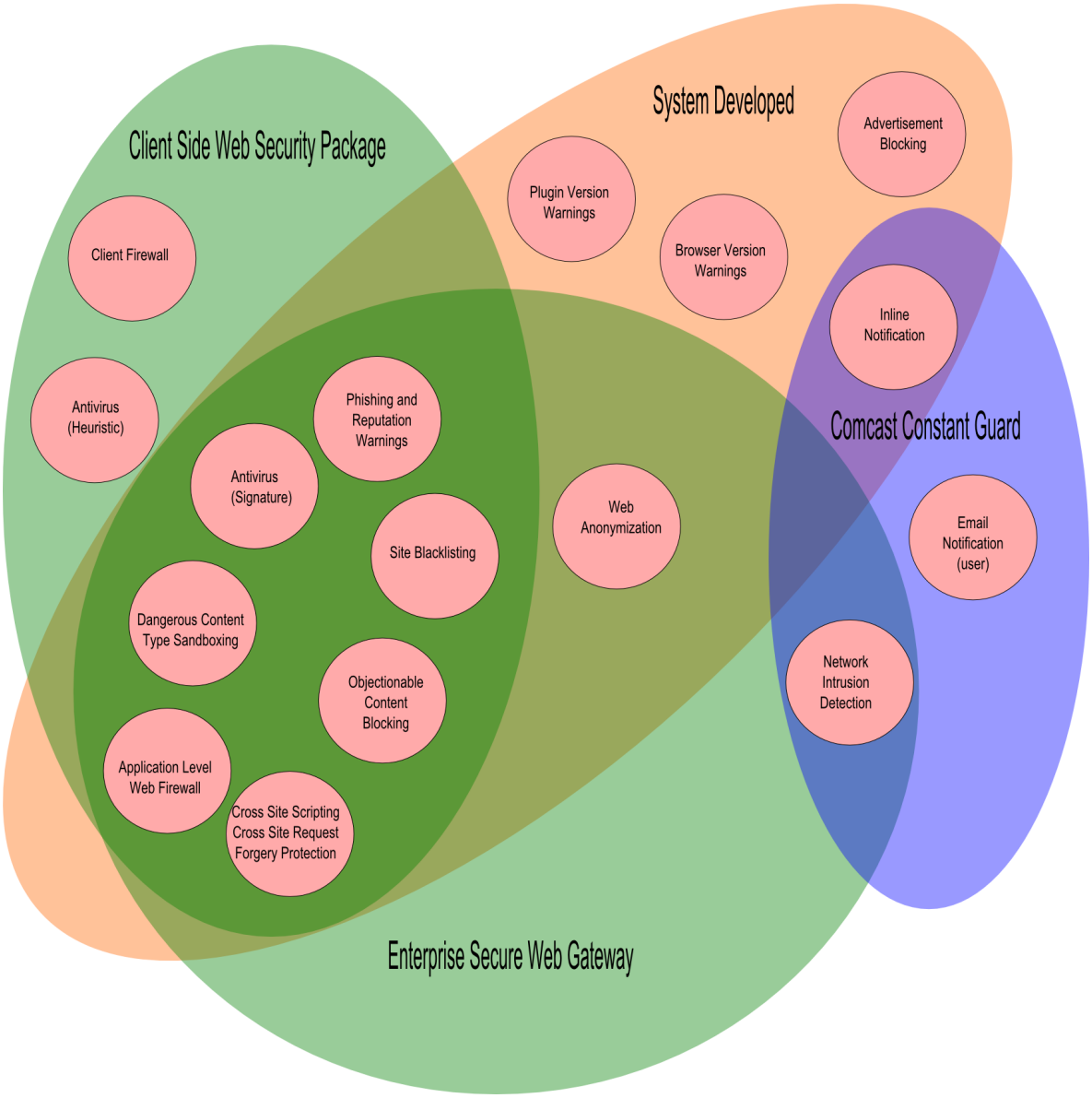

Figure 3.6: A Venn diagram of the proposed feature set and some other solutions

3.9 Specific Operations Implemented

The security operations performed fit into three main feature groups: user messaging (from a database), inline warnings, and content scanning and filtering. Of the scripts developed, some are included in the appendices for background (Appendix H), however we will discuss the operation of each major script developed briefly next.

Type	Implemented Functions
User Messaging	Global Userbase Messaging User Specific Messaging Rule Based Messaging
Inline Warnings	Browser Version Plugin Version
Content Scanning and Filtering	Site Blocking and Blacklist Checking Keyword Based Blocking Reflective XSS Filtering User-agent Anonymizer Content Type Filtering Content Type Redirection Remote Antivirus Scanning

Table 3.2: The Greasyspoon scripts implemented

3.9.1 User Messaging From a Database

While it is possible for a script to perform message insertion with no external store for the data, it is not very flexible or extensible. Therefore we developed scripts that check the database and display the appropriate messages from there. (There are some as yet unresolved concurrency, performance and caching issues, as discussed in the results.)

Global userbase messaging

This script searches for the body HTML tag, and if it finds it it inserts the message code. The message code used in testing includes that needed for an HTML layer element (DIV tag), CSS to format the layer, and simple Javascript to enable a link that minimizes the layer to the bottom-right page corner or shows it in full. The code inserted is given in appendix H.1.1, but the effect of its use on the TradeMe website under Chrome is shown in figure 3.7.

This was designed to appear independent of the rest of the page content, and does not affect the flow of content on the desktop browsers we tested it on (Internet Explorer, Chrome, Firefox, and Safari). Figures 3.8 and 3.9 show how underlying content remains unaffected when the message is minimized. The effect on pages viewed on mobile devices is unknown, but it should be trivial to modify the insert surrounds to suit different client browsers. The insert used could also be a problem on pages that have a crucial element in the bottom right, but this space was chosen because most websites have top or left navigation, so it is unlikely to be an issue. Should it be desired, it is a simple matter to change the minimize functionality to hide the insert completely.

The content and formatting of the message inserted are trivial to modify, however at present we can only insert messages into HTML pages which contain a body tag, and cannot easily

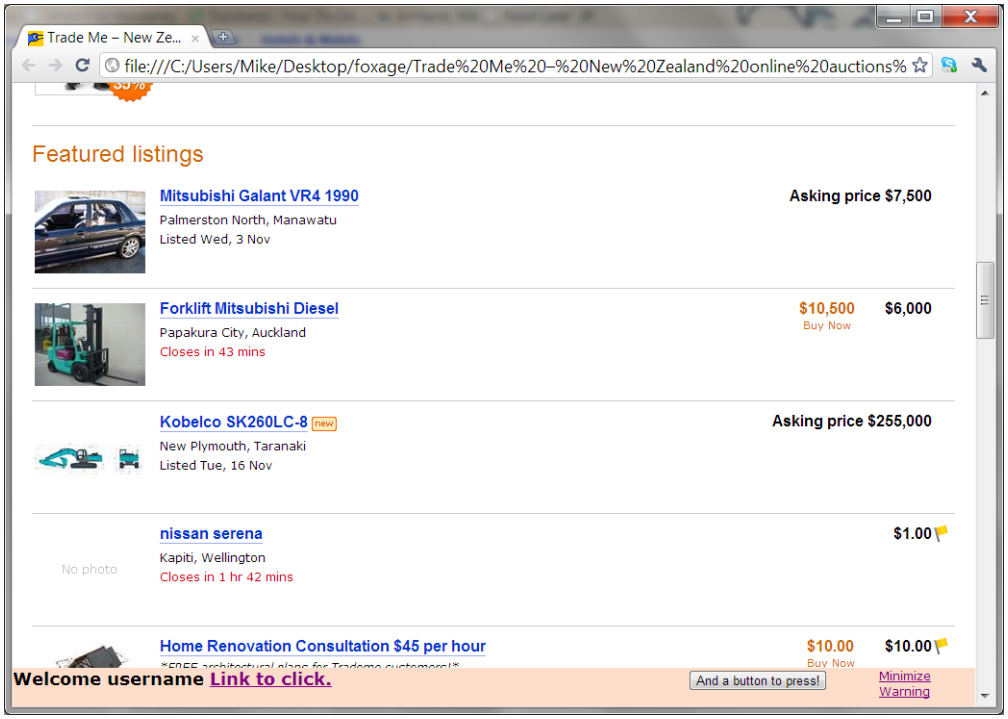

Figure 3.7: An example insert shown on trademe.co.nz

discern the main page from any smaller partial pages such as advertisements in iframes[4]. Furthermore, although not unexpectedly, some pages display strangely with the inserted code for various reasons.

Rule Based Messaging

Rule based messaging first checks whether some condition is true before inserting the message. This condition is arbitrary, but in testing corresponded to a condition such as something to do with the content or URL of the page being accessed.

User Specific Messaging

User specific messaging is a specific sub-group of rule based messaging that checks the accessing user's details against a database entry and displays a message specific to that user. The user's identity can be assessed by checking the proxy login (as sent in the ICAP header), or by looking up the IP address in the database. Although functionality such as the ability to dismiss a message or set an expiry time could be implemented simply enough, it was not felt to be necessary for this project.

[4] An iframe is a small inline frame that displays one HTML page inside a section of another.

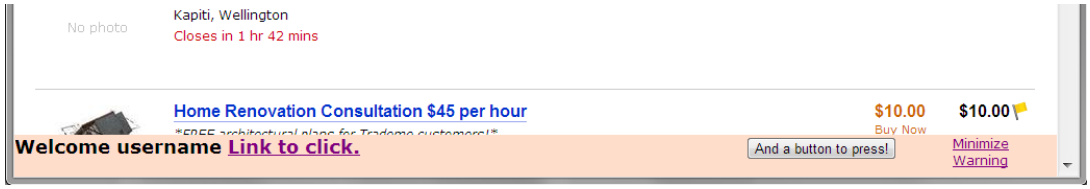

Figure 3.8: The insert in non-minimized form

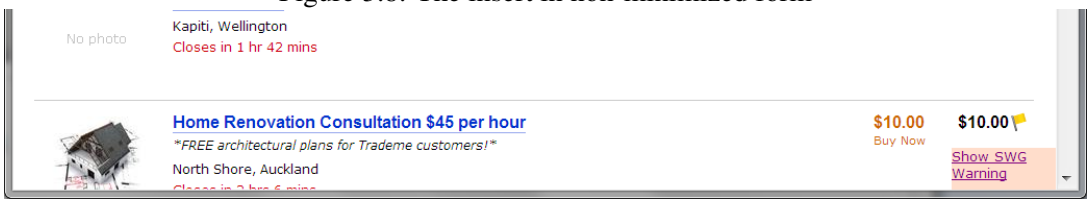

Figure 3.9: The insert minimized

3.9.2 Inline Warnings

These scripts display warnings to users based upon something they see in the traffic - they are more specific examples of the rule based messaging detailed previously. The operations used in this section are all what we term pseudo-stateful in that they require the coupling of request and response scripts to function. Generally this will involve making some observation on the HTTP requests, and triggering some change or alert in the HTTP response if necessary. Some of these operations also allow the user to bypass an initial warning that blocked them from accessing a resource.

Although some of these scripts make use of the messaging functionality, we consider them a distinct part of the security solution because they are contained entirely within it. It would be possible to emulate some of the functionality here through the use of an Intrusion Detection System (IDS) to inject messages into the database - we discuss this in further work (section 7).

Browser and Plugin Version Warnings

Outdated versions are a major source of security vulnerability for drive by downloads, as well as a problem for site display and general usability (as discussed earlier). Many users are, however, unaware of this, and may be completely unaware that they are running an outdated browser. This problem may be exacerbated when users reinstall from backup, system restore points, or do a clean reinstall – as many automatic update systems only come by default with later software[5].

The user-agent HTTP request header may contain a lot of information on the software versions of the accessing system. The details included in the user-agent can include the versions of

[5]For instance early versions of Windows XP either do not have automatic updates through Windows update, or have it turned off by default.

the client operating system, web browser, and sometimes also available browser plugins. While it is not necessarily practical to recommend users upgrade their operating system, upgrading the browser and plugins is a very important thing which can generally be done by running a single download or updater.

The User-Agent sent in HTTP requests differ in the level of detail between various web browsers, however, the structure generally remains the same. Lately many web browsers have taken to impersonating the user-agents of other browsers.[6] For instance, on the system on which this document was written four browsers all gave "Mozilla/5.0" as the beginning of the user-agent they supply to sites.

- Internet Explorer 9 Beta
  - **Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)**
- Firefox 3.6:
  - **Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 (.NET CLR 3.5.30729; .NET4.0E)**
- Chrome 7:
  - **Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.7 (KHTML, like Gecko) Chrome/7.0.517.44 Safari/534.7**
- Safari 5:
  - **Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.2 Safari/533.18.5**

Not all web browsers supply Mozilla/5.0 at the beginning, but irrespective it is possible to deduce the exact browser version from other parts of the text anyway. Furthermore, in the case of Firefox two versions of the .NET framework can be seen.

In the system developed, the functionality to warn users about outdated browser and plugin versions is a request script component which works in conjunction with the user-messaging script to alert users. The request script at present requires outdated browser versions to be defined and listed manually in the request script. To make the user-alerts functionality, the user-agent string is scanned, and if a blacklisted version is detected, a message is inserted for the user.

[6]There are plenty of lists of user-agent strings online, for instance http://www.useragentstring.com/pages/All/

This functionality is perhaps oversimplified for users who use more than one device, as they will then potentially receive the message on all their devices – for instance if their laptop is running an outdated but their desktop is not, they will receive the message on both anyway. Methods exist to work around this, such as putting short timeouts on the message (which won't work if the user is using devices concurrently), or displaying the message only in response to specific user-agent bearing requests, but these add complexity and are not within the proof of concept scope of this project.

3.9.3 Content Scanning and Filtering

The scripts here examine something about a request or response and then perform an operation. This can involve examining HTTP headers, textual content, or binary content of requests and responses. If some condition about them is met then various operation are performed, such as a message being inserted into the database for the user, the content being modified in some way, the user being redirected, or the content being blocked completely.

Site Blacklist Checking and Warning

Site *blacklisting* (as discussed in section 2.7.2) is a method whereby sites are checked against a list of disallowed sites, and if they are on the list then access is denied. Blacklisting is useful for protection against malware, fraud, and potentially Data Loss Prevention (DLP)(for instance, by blocking external email) and organizational IT policy (for example banning social networking sites in a workplace).

Although they only work against known bad sites, and are useless against new rogue sites, blacklists can nevertheless be a useful security measure. This is particularly true if a good regularly updating blacklist source is used and the lists used are themselves regularly updated. Whatever the motivation for their use, three types of blacklisting were implemented in this project: static blacklists, automatically updating blacklists, and blacklist services. When an attempt was made to access a blacklisted site the request was not serviced, but an HTTP response warning page returned directly to the client. Note that the address in the URL bar is the same as the site requested, because as far as the browser is concerned it has received the page it asked for.

If the user wishes to bypass a blacklisted page, they can override this block by clicking the override link on the blacklisted response page. This is handled by allowing the user to access a forbidden site if there is an additional string of text at the end of the URL and the referrer of the request is the same as the site being requested. The referrer is there because this is a rudimentary

way to ensure that the user clicks the link on the response page, and didn't fall victim to a CSRF request.

For example if "http://www.google.com" is blocked and our bypass text is "&&bypass=true", then the override link on the blacklisted response page will point to "http://www.google.com/&&bypass=true" and will function only if our referrer is "http"//www.google.com. This override treatment is far from perfect, and is still vulnerable to some forms of attack. For brief discussion of methods we believe may improve the security of the bypass functionality, see section 7.2.1.

Static and Automatically Updating Blacklists In the static blacklisting script, a predefined string array contained blacklisted domains that the request URL was checked against. This functionality was present in the Greasyspoon default example scripts, but we slightly modified the way it worked. In the provided scripts the sites to which it was applied were defined by the Greasyspoon script "@include" and "@exclude" filters, whereas we moved the blacklist from there into the script contents to that it could be programmatically updated and checked.

Although the script used had sites defined initially, they could just as easily be read from a file or database during program initialization[7]. Programmatically updating the list also adds the advantage of enabling updating the list while the program is running, although this was not done successfully due to concurrency errors that were not overcome (as discussed in Results (chapter 5) and Future Work (chapter 7).

Blacklist Database Lookup used a relational database which is used to look up the current URL in a list of blacklisted sites rather than storing the list programmatically in a data structure. If the URL is found, then the site is treated as blacklisted. Using a database rather than statically coding it or using a flatfile has some advantages. A database is designed with concurrency and speed in mind, so rather than the Javascript needing to do many comparisons this is handled by the database server. The use of a database also considerably simplifies updating blacklists, as this can be done by an external script, with concurrency and locking handled by the database system.

The Google Safe Browsing API library used (jgooglesafebrowsing) uses this method, except that rather than looking up the site name directly the MD5 hash of a *canonicalized url* (an URL

[7]This could cause significant performance problems however due to the way Greasyspoon loads and unloads scripts with each request. The lists can be stored in Greasyspoon's Java shared cache which uses hash based data storage and persists across the execution of multiple scripts.

converted to a standard format, see [58]) is used so that the blacklist cannot be looked up in reverse to generate a list of blacklisted sites. Update functionality is included with the library in the form of a java function call, we ran externally to the Greasyspoon script, because if included in the Greasyspoon script it blocked the call that triggered it until the update was complete, and if the update took longer than the Greasyspoon script timeout, then the update never successfully completed.

Third Party Blacklist Services Using a similar method to the database lookup script, except that the lookups are handled by a third party. The lookup can be done through various forms including network API's, DNS lookups, or web services. We used a DNS lookup service, Spamhaus's Exploits Block List (XBL) [162], where blacklisted sites return bad values indicating the reason for blocking. The DNS lookup method was used because DNS lookups can be undertaken in Java, and can be very fast in response.

Keyword based blocking

Although not strictly a security operation, blocking or censoring pages based upon keyword contents is an operation which may be desired in some contexts to control user browsing. The script to achieve this functions in a similar way to the static blacklisting used for page URLs, but rather than checking the page URL against the list, it checks whether the page body is HTML, and if it is whether it contains any of a predefined list of words. If any blocked words are contained then an erorr page was displayed (as for blacklisted pages), but it could be simply modified to insert a message for the user, alert an administrator, or similar. The list of blocked words is treated in the same way, and raises the same issues as with static blacklisting above.

Reflective XSS Filtering

A simple method of protecting users against reflective XSS attacks is to encode or block any special characters or keywords that are contained in a GET request [8]. For example, if the user requests *http://www.google.com/search?q=<script>...</script>* then something strange is probably happening.

The XSS prevention script looks for the < character and many of its encoded variants (using Rsnake's well known list of 81 versions at [142]), and if it detects it then the request is blocked

[8] Filtering post requests is more complex, as there are many situations where a user may legitimately need to send scripting content to a server. An example of this would be if they were using an online HTML/JavaScript editor

and a warning page returned. Although this is a simplified attack detection mechanism it should detect most types of XSS attacks that use the < character. Some types of attacks do not use this character though, and as such the script developed does not protect against them. We briefly discuss how this script could be expanded to offer protection against a broader variety of XSS attacks in section 7.

Content Type Detection

Section 2.2 discussed the security of various content types, and how some types of web content present larger security threats than others. Among the dangerous types at present are SWF, PDF, and Java content. While blocking SWF may cause many sites to stop functioning correctly and is probably best left unhindered, PDF and Java content do not normally need to be loaded without user intervention and so security can be improved by intercepting them and warning the user that they are about to be loaded or redirecting them to an online viewer (see section 3.9.3 next).

There are three main ways we use to detect the type of web content, with the reliability of methods correlated with difficulty. In order of difficulty, detecting the content type of a file can be undertaken by checking one of: the file extension requested, the MIME type returned by the web server, or some signature characteristic of the filetype.

File extensions refer to the part after the period character (.) in the filename, and are the the most efficient guide for the content type of a file. For instance .htm, .html are HTML files, .exe indicates Windows executables, .pdf .doc .docx each indicate various document formats. The use of file extension can cause problems however, as sometimes the extension is not useful at all to determine the filetype. Ambiguity can also be an issue, as some server side scripting languages have file extensions to indicate the script type to the web server, and the content they send to the client can be of any type. For instance a .php script may generate and send a .pdf file to a client. Furthermore, there is nothing to stop a file being given a deceptive file name to intentionally bypass filters. To overcome some of these problems the client can look at the file MIME type as sent by the web server.

File MIME type is sent in the HTTP headers of an HTTP response to indicate to the client how to handle the transfer and display of a file. Transferral of a file may depend on whether the file is in a text or binary based format (for instance 7 bit based ASCII text, 8 bit based binary, or 16 bit based Unicode), and if the client expects a binary and receives a textual file (or vice versa)

file corruption can occur.

MIME type is used by some web browsers to decide how to handle a file, for instance whether to display it as an image, render it as text, run it as a script, or send it to a plugin. Web servers may determine the MIME type of a file from the file extension (as above), or they may look it up in a so called *magic number* file, which contains lists of characteristic bytes that show up in each file type. We discuss this in more detail in file signatures next. A web server is under no obligation to send the correct MIME type, however, and a malicious server may send the wrong type and thus deceive a client application into mishandling the file.

File Signature File types usually contain some sort of pattern in how they store their data, and usually there are structures in the data structure that can be used to assess the type of file being analyzed. These structures which can be detected are called file signatures, and often (but not always) consist of a set of header bytes at the start of a file. The binary representation of these bytes is known as a *magic number*, and a file containing a list of magic numbers and the corresponding file types is known as a magic number file. Unix machines contain the *file* application that can be used to assess the file type of a file via a magic number file in /etc/magic.

Greasyspoon is supplied with its own magic number file, and when the magic.mime setting is enabled it uses it on incoming files to determine the MIME type. While they could rely upon this Greasyspoon functionality the scripts used in this system tend to only be looking for a specific filetype, and so check the file contents themselves.

For instance the PDF detection script looks for the following characters in the header of a file to detect a PDF file:

- **%PDF-** (ASCII Text)
- **37 80 68 70** (Decimal bytes)
- **25 40 44 46** (Hexadecimal Bytes)

Once a relevant filetype is detected then the appropriate action can be taken, whether blocking the file or redirecting the user as we do next.

Filetype Redirection

Much of the security risk around client side web browsers involve improper filetype handling on client software. Some client software can be bypassed through the use of free services online

that do not require the user to to receive the file at all, but uses server side rendering to display a file retrieved directly from the web server in an online viewer.

The service used in this project is the Google Docs viewer [60] which renders the contents of document files in the PDF, DOC, DOCX or PPT formats without loading the file contents (or any scripting content) on the viewer's machine. Fig 3.10 shows the Google docs viewer which document types were redirected to during security testing, although other services such as the Zoho docs viewer [194] or Samuraj Data AB's viewer [144] could be used. Alternatively, the organization could install software such as Adeptol's server based viewer software [2] on it's own server and use that instead. The use of these services should prevent much PDF file format based malware from affecting client machines, as the rendering and scripting engines are run remotely, and these services do not normally support embedded files or scripting.

Should the user wish to download a file after viewing it in the Google Docs viewer, this was enabled through the use of a HTTP referrer check; if the referrer was docs.google.com then the file is not redirected, and the user receives the file directly. The user must view it in the viewer first however, preventing many automatically loading exploits from affecting the user.

Antivirus Scanning

Antivirus scanning was undertaken with a script that sends all files the ICAP server receives to a remote ClamAV antivirus scanner service listening on a TCP port (either remote or on the local machine). The file is downloaded to the proxy, which then sends it to the ICAP server, which then forwards it to the antivirus server and reads the response before telling the proxy to send the content unmodified or send the error page. The script written was supplied to the project lead, and is now available as an example script on the Greasyspoon web site at [61], and is also supplied in Appendix H.2.1.

The anti-virus script functions by connecting to the Anvitirus server and sending the STREAM command. It should then receive a reply containing a TCP port number to send the file to, a second connection is opened to which the file is sent, if a virus is detected then FOUND is output to the first connection, as well as the virus name which was detected. In the event of a detection then the file is blocked and a warning page is displayed. This is illustrated in fig 3.11.

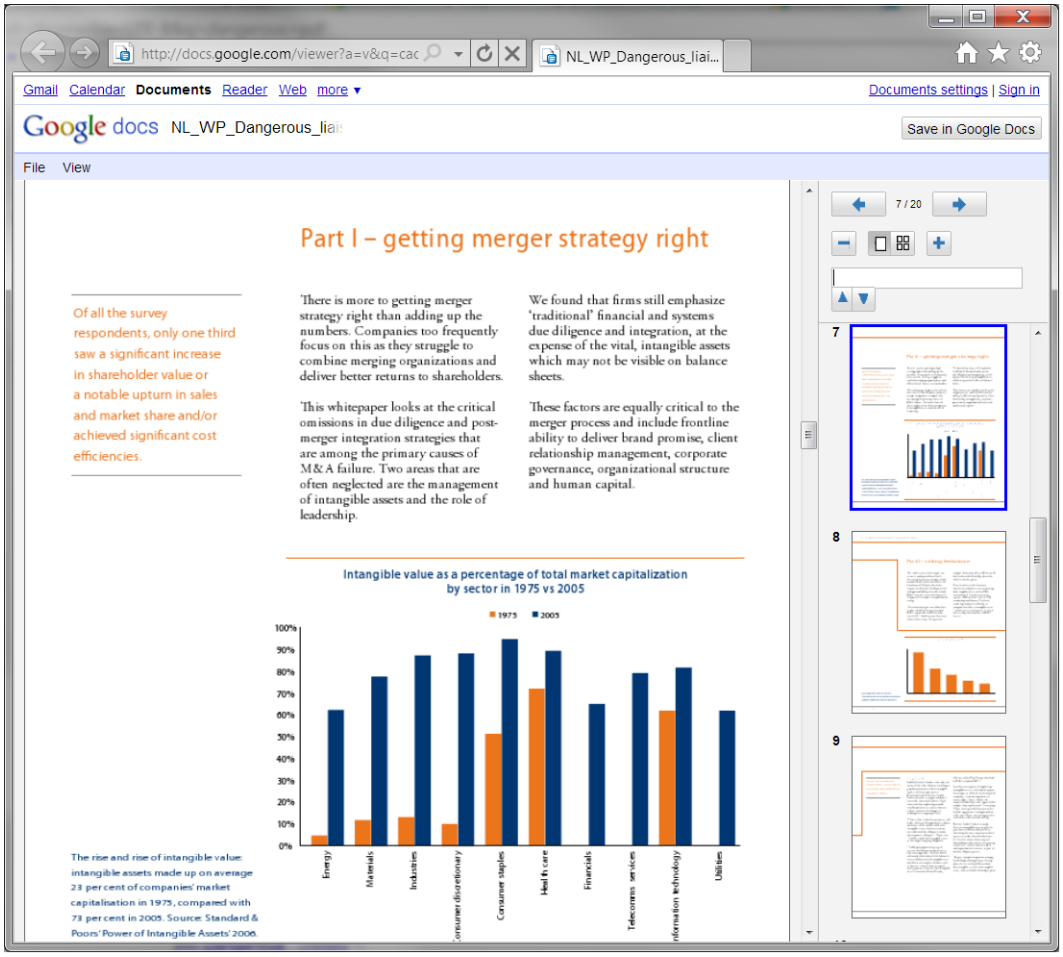

Figure 3.10: A remote PDF file being displayed in a web browser inside the Google Docs online viewer

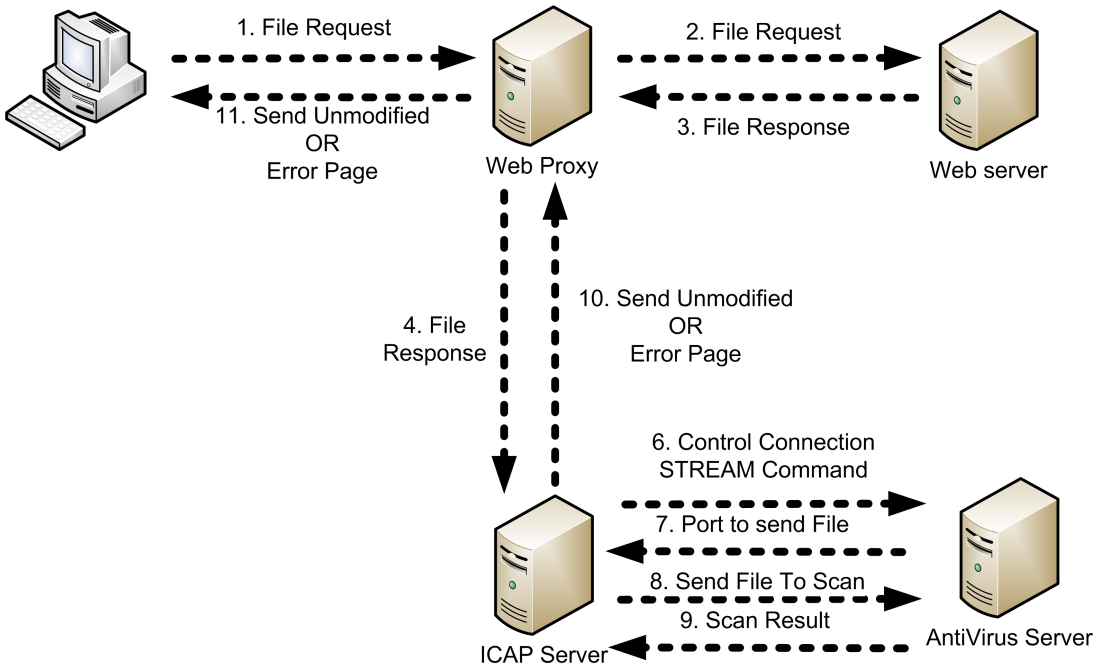

Figure 3.11: Order of operations for the antivirus script

While suitable for testing, this has a few implications for a production rollout: the file must be transferred across the network four times, adding latency which may lead to a client timeout; files larger than 10MB are skipped as they will cause this timeout too often to be useful. Although the antivirus engine used was ClamAV, any ICAP or TCP enabled engine should be able to be used with a few modifications. This particular scripts works for a proof of concept scenario, but should be refined for use in a production environment.

Chapter 4

Testing Methodology

While developing the architecture and operations for an open source based ICAP secure web gateway add to the body of knowledge, it is important to know if the solution shows any promise for real world application. Two main aspects were judged as important to assess the usefulness any network security service such as this: effectiveness against threats, and performance impact upon users.

4.1 Testing Overview

Three main types of testing were undertaken for the system developed in this project: inspection to verify that simple operations functioned correctly (such as user messaging), automated comparative testing of effectiveness against both unknown and known malicious datasets, and comparative performance testing of representative script types in a variety of load conditions.

4.1.1 Qualitative Verification

All simple script operations (such as those for user notification) that performed very simple or trivial operations were verified manually, mostly while the scripts in question were being developed. Verification was undertaken for each script developed, and examples of it include checking that sites on a blacklist were blocked correctly, messages were displayed with the correct user or users, and that modified data was sent across the network correctly. Early issues and enhancements for many of the scripts were identified at this stage also. The results of this are discussed in the next chapter.

4.1.2 Quantitative Verification

The testing of effectiveness and performance was undertaken using a methodology that gave quantification of results, and comparisons with various controls. Effectiveness testing was performed by using four sets of client honeypots (as discussed in section 2.4.3), with each set visiting the same sites as the other sets, but utilizing a different protection measure. By comparing the difference in infection rates between the sets of honeypots, the comparative protection was assessed. Effectiveness testing is discussed in detail in section 4.2.

Performance testing was also undertaken in a comparative way. The time taken to service requests to an internal web server was measured using ApacheBench under different load situations. Performance testing is discussed in detail in section 4.3.

4.1.3 Quantitative Testbed Overview

Because the system developed requires all web traffic to flow through the web proxy server, any testing architecture may connect to the proxy to use it. Two different testing architectures were used, one that was used to test performance, and one to test effectiveness. The testbed was composed of multiple PC machines with identical hardware configurations [1] connected by Gigabit ethernet and connected to the Internet via a gateway that allowed outbound connections via Network Address Translation (NAT).

The systems that were used in the performance and effectiveness testing were set up in a dual-boot configuration, with one operating system used in the effectiveness setup, and another in the performance testing setup. Figure 4.1 shows the entire testbed (including third party Internet servers used) and the logical connections between them. Details on the effectiveness and performance testing methodology and setup are discussed in sections 4.2 and 4.3 respectively.

The network details for the full components are given in Appendix B.

4.2 Effectiveness Testing

The effectiveness testing methodology used is believed to be novel. Testing was undertaken using client honeypots (Capture-HPC version 3.01), automated web browsing virtual machines that visit lists of sites and perform integrity checking to check for traces of malicious actions while, and after, visiting each page. The effectiveness testing system discussed here crawled around 5 000 pages per day for each protection measure, for a total of around one hundred

[1] Core2 CPU at 2.2GHz, 2GB RAM

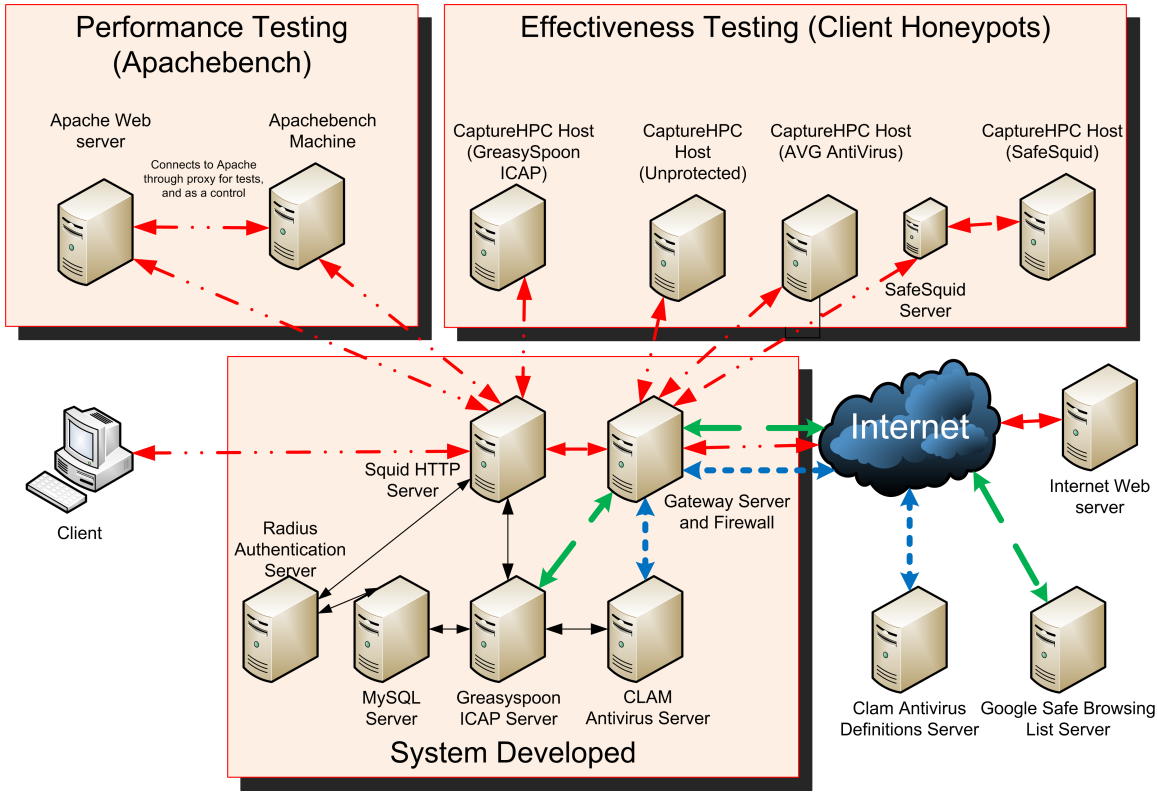

Figure 4.1: Our testbed components

thousand web site requests over the course of the effectiveness testing. Further details on the testbed's operational conditions are given in the next chapter.

4.2.1 General Methodology

To gain a comparative view between effectiveness of the system developed and some alternatives including unprotected systems four sets of honeypots were set up to visit the same sites in the same situation in parallel with the only difference between them consisting of the protection measure used.

4.2.2 Honeypot Integrity Checks

The software component of Capture HPC (version 3.01) installed in the honeypot virtual machines monitors the following aspects of system behaviour:

- Windows Registry keys and values - reads, changes, and deletions
- Network ports and connections - opened or started listening
- Files - reads, writes and deletes
- Processes - started and stopped (and the parent or terminating process)

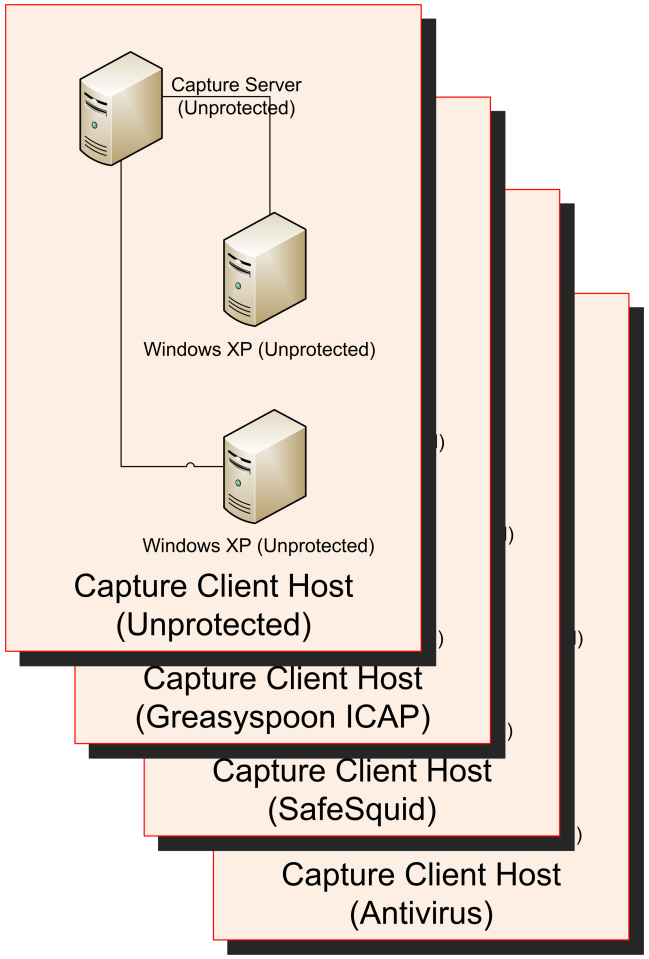

Figure 4.2: Capture-HPC server layout

All operations detected were compared with a whitelist of acceptable operations, and if any non-whitelisted operations occur, the details were logged. The capture client could also gather any files that are changed and capture network traffic. The logged details and files were then forwarded to the server, and the virtual machine was reset to a clean state before continuing.

4.2.3 Comparative Detection Measures and Controls

Honeypots utilizing four different levels or types of protection mechanisms were used: Unprotected, AVG free antivirus, Safesquid, and the system developed. The use and motivation for the use of each is briefly discussed below.

The ICAP System developed, and unprotected systems were used as one is the system we wish to evaluate. The unprotected systems were used as a control so that any difference in effectiveness over an unprotected system could be measured.

AVG Free Antivirus was used to represent client side antivirus protection measures. AVG was chosen because at the time of preparing the testbed it appeared to be the most commonly used free home antivirus system that offered web browsing protection. It is difficult to deduce market share of such things as vendors do not seem to readily disclose such facts, so the metric that was used was the number of downloads in the previous month from the major software website download.com.

SafeSquid was chosen to represent the proprietary web security proxies. Safesquid was chosen because it has a similar feature set to the system developed, is widely deployed, and has a downloadable version for small deployments (up to three users) that could be easily accessed.

4.2.4 Client Honeypot Configuration

The configuration of the client honeypots is documented here so that the attack surface exposed to threats can be understood. An overview of the software installed in the honeypots is given in table 4.1, and further details are given in the rest of the section.

Software Type	Details
Operating System	Windows XP Professional SP0 with SP2 Installed (5.1.2600SP2)
Web Browser	Internet Explorer 6.0.2900.2180.xpsp_sp2_rtm.040803-2158
Software with Browser Plugins	Flash 10.0.45.2 Quicktime 7.1.5.120 Java 6.0 (update 16) Acrobat Reader 8.1.1
Other Software	Capture-HPC Client Software (3.01) VMWare Tools (VMWare server 1.09) Clickoff 1.76/1.82 WinPCAP AVG Antivirus 9.0.819 Virus DB 271.1.1/2967

Table 4.1: The software installed on the client honeypot

Virtual Machine Settings

The virtual machines were set up in VMWare server 1.09 with 384MB of memory and one virtual CPU allocated to each VM. All VMs were cloned from the same source VM after all configuration other than that needed to separate protection mechanisms (proxy settings, installing AVG etc). During testing four VMs were initially run concurrently on each physical machine, however by later test runs this was reduced to two per machine to reduce errors. Windows on

each virtual machine was logged in with an administrative account (as users commonly are on Windows XP) and it's inbuilt firewall was also disabled.

Finally, a VM snapshot was taken after the VM operating system had been started and left for a few minutes, this allowed the snapshot to be taken after any background processing and IO operating had ceased. This was the snapshot which Capture-HPC restored regularly during testing.

Modifications from Default Behaviour

All forms of software updating (except antivirus definitions) were disabled to keep the versions consistent across test runs. This included Windows update, and auto updating of Flash, Acrobat Reader, Quicktime and Java. Windows update was disabled through the windows update settings graphical interface. Flash automatic update was disabled by modifying mms.cfg using the method at [3], Acrobat Reader updating was disabled according to the instructions at [4], while Quicktime and Java automatic updating were disabled in the appropriate settings menu.

Internet Explorer had the security settings applied to Internet zone sites changed because too many dialogs were being triggered and hampering testing. The settings were changed from medium security to low security, exactly as a user annoyed at the same dialogs might be inclined to do. Asian language packs such as Chinese (Mandarin) were also installed so that websites in non-western scripts could be rendered.

Clickoff [74], a tool that automatically simulates a predefined click on a pop-up dialog option, was installed and configured to automatically click OK or Continue on warning dialogs, and to automatically attempt to run any downloaded files. This was done to simulate a naive user (potentially a child) that ignores all Windows security dialogs. While this decision potentially resulted in more infections, it does not bypass any security warning pages generated by our scripts, and so it enables the assessment of protection against malicious downloads that require the user to run them.

Reasons for software versions chosen

Versions were chosen to represent a computer system with out of date software. This was done because it would firstly increase the number of attacks the system is vulnerable to, and secondly

it would enable assessment of the level of protection offered to systems with few client side defenses.

- Windows XP Service Pack 3
  - Still used by many users, vulnerable to many types of attacks with few client side protections compared to later Windows Versions
- Internet Explorer 6
  - Still used by some home and corporate users, vulnerable to many types of attacks with few client side protections compared to later Versions.
- Adobe Flash
  - Version a few months old and vulnerable to a few in the wild attacks. Flash automatic update is quite difficult to disable in the version used, so unlikely to be much older.
- Adobe Reader
  - Difficult to find old versions, used very old version. Not problematic for this software as old versions still come with software and driver cd's (for reading manuals).
- Quicktime
  - About 3 months old, had weaknesses being exploited in the wild at the time.
- Java
  - Not unreasonably old (about 3 months and 3 versions old), had weaknesses being exploited in the wild [2].
- AVG Antivirus (and definitions)
  - Updated software and antivirus definitions before each test run then took a fresh snapshot.
- Capture-HPC client, VMware tools, Clickoff, WinPcap, AVG Antivirus (and definitions)
  - Utility software, installed latest version available at the time and updated antivirus.

[2] The Java Web Start Vulnerability - CVE-2010-0886 [34]

The use of deliberately obsolete versions of software may have skewed the experimental results. Future work 7.3 includes the use of more varied and comprehensive client honeypot configurations including updated systems that should result in improved testing applicability with real world protection.

Capture-HPC configuration

The Capture-HPC settings used were refined iteratively over several tests. Initially they were far higher than the final figure but problems with errors caused by exceeding system resources required these to be lowered. The final settings used are given in Table 4.2, and the full configuration file is given in Appendix F.1.1.

Setting	Explanation	Value
Visit Time	Time spent visiting each group before moving on to next	90 Seconds
Group Size	Number of pages in each group	5
VM Stall After Revert Timeout	How long to wait for reply after snapshot revert before assuming a crash	180 Seconds
VM Revert Timeout	How long to wait for a successful snapshot revert	120 Seconds
Client Inactivity Timeout	How long unresponsive before crash presumed	120 Seconds
VM Stalled During Operation Timeout	How long unresponsive before crash presumed	120 Seconds
Same VM Revert Delay	Minimum time between different reverts of a VM	12 Seconds
Different VM Revert Delay	Minimum time between reverts of any two VMs on same machine	12 Seconds

Table 4.2: The Capture-HPC server settings in config.xml

4.2.5 ICAP System Configuration

The system architecture discussed in 3.5 was used as protection measures for one of the honeypot systems. The configuration is generally as described in section 3.6, with the scripts given in table 4.3 used initially[3]. Some of these were disabled after test runs, as discussed in section 4.3.

[3]Scripts which performed operations targeted at notification were not used during testing of automated measures.

Type	Script Operation	Settings
Request	XSS Filter	$<$ character blocking
Request	Google Safe Browsing	Flatfile not database. Disabled after preliminary test runs due to problems (discussed in results section 5.1.3)
Request	Redirection to Document Viewer	Google docs viewer for pdf, ppt, doc, docx, pptx.
Response	File Signature Blocker	Blocks pdf files by signature
Response	Antivirus Scanning	Remote ClamAV Server

Table 4.3: The automated response scripts used in effectiveness testing

4.2.6 URLs Used

There are no security operations performed that are in any way novel or controversial, but it is important for a system such as this to be tested for effectiveness against real world attacks. This is because, due to the rapid speed of this type of attack, many of the attacks encountered will be novel, use Zero Day Vulnerabilities, or be hosted on Fast Flux Network[4] networks. Testing was undertaken on Internet URLs from one of two URL lists: known malicious URLs taken from a 3-day delayed blacklist, and automatically gathered URLs from online trends gathered hourly while testing (to ensure active and current threats).

Both unknown and unknown datasets were used for four main reasons: the unknown set was used to provide a higher rate of novel and active attacks, whereas the URLs from the blacklist were used to act as a control, to increase the overall number of malicious sites encountered, and to test effectiveness against attacks for which antivirus signatures could reasonably be expected to be released.

Unknown / Harvested URLs

For the unknown data set large numbers of URLs were harvested from the social networking site Twitter through the use of a set of Python scripts (given in Appendix F.1.2) that searched for links mentioned alongside popular (or *trending*) topics.

Motivations Three main reason led to the decision to use URLs gathered in this fashion: the need for a realistic sample of real world threats, the need to sample effectiveness against fast

[4] A fast flux network is a network that is hosted upon or proxied through a fast changing arrangement of compromised hosts.

moving threats which quickly change location or attack mechanism, and the need to sample threats that make use of *black hat SEO* (as discussed in section 2.7.3).

Twitter search was used because it is less filtered than the listings on a search engine such as Bing or Google [5]. A large proportion of the links gathered were from short URL services (see section 2.4.3), some of which are filtered anyway (Such as bit.ly through its partnerships with Verisign, Websense, and Sophos [170]), nevertheless malware links were found in the URLs gathered (discussed in the results section 5.3) next).

The List Building Script (given in Appendix F.1.2) was written in Python and used the following procedure to gather URLs which were then fed to the honeypot. [6]

1. Collect trends into a list (Twitter instant (current), Twitter daily, Twitter Weekly, Google search current)
2. Remove duplicate and non-textual (such as trends in Asian character sets) trends from list
3. Search Twitter for messages which use the terms in the trend and the term "http"
4. Extract URLs from the results (using a regular expression)
5. Append URLs to file.
6. Wait 1 hour, then start again.

Duplicate URLs are not removed, as Capture-HPC handles this automatically.

Known Bad URLs

A set of URLs known to be high risk or to contain malware was used in effectiveness testing because it enabled us to test the effectiveness of the system against targets which were of the type the system developed was designed to protect against. Furthermore, given the relatively low rate of malicious sites in the unknown data set (less than 0.5%) the use of this dataset enabled testing against a larger number of malicious sites, thus improving the likelihood results were representative.

The blacklist used was the free list from http://www.malwareurl.com, which was 3 days delayed and from which the top 1000 URLs were used. While the paid version of the blacklist is not delayed, it was not felt to be a problem for the testing we undertook. However, this delay did

[5] Google's JavaScript API was used in initial testing, however it was discovered that it removed malware sites completely from the listings returned, even if they show up on the normal search page with a warning of "this site may harm your computer". This could be overcome easily by scraping the page, but it was ultimately decided that Twitter would provide much quicker moving results.

[6] Debugging and information text is also inserted into the file, but is not detailed here.

result in high error rates for the URLs in this list, as many of them were on compromised hosts, had been taken down by the provider, appeared to be on fast flux networks, or were otherwise inaccessible. Those which were still accessible did have high malicious rates.

The delayed nature of the blacklist used also allowed a comparative between the signature based methods (antivirus) used in testing and the non-signature based methods used in the system developed.

4.3 Performance testing

While effectiveness testing was undertaken against alternative systems, performance testing was undertaken to work out the performance characteristics of the system developed. This approach was undertaken because usually inadequate performance can be augmented with software optimizations or more hardware, and we desired to test the system developed and not the underlying hardware/software test base. Performance characteristics were tested to ascertain the following, and the interaction between them:

- System performance of different modification operations
- System performance under different load conditions

The remainder of this section discusses the testing undertaken to ascertain the general characteristics around these.

4.3.1 Architecture and Tools

Testing was undertaken using the web server benchmarking tool Apachebench[168], and an Apache web server (version 2.2.14 from the Ubuntu 10.04 repositories) running on another machine on a Gigabit ethernet link.[7] Apachebench automates requests to a web server and measures characteristics of responses, including time taken to fully service requests (broken down by percentile). For the testing undertaken in this project Apachebench was configured to send the requests through the proxy server, thus testing how request delay is affected by the use of various elements of the system. This is illustrated in figure 4.3.

4.3.2 Variables and Controls

Twelve cases were used to determine comparative performance of different operations and scripts developed. To test each, five different types of variables were used to test the characteris-

[7] The Ethernet link was tested using Netperf [79] and had a throughput of 111.66MB/s between the machines.

tics under load. Thus each of the twelve cases was tested through one hundred load conditions. This was performed in an automated fashion (using the Linux shell script given in Appendix F.2) for a total of one thousand two hundred test runs comprised of a total of around six-million requests made from the web server.

Representative Script Classes

While Greasyspoon is used to perform many types of operations in this project, the operations it performs can fit into several main types. We also use various controls to ensure that the testing is hitting a limit of the system developed, and not hitting some bottleneck in a system component. The script classes used are given in table 4.4.

Test Case	**Scripts Used (if relevant)**
No Proxy, No ICAP, No Scripts	
Proxy, No ICAP, No Scripts	
Proxy, ICAP, No Scripts	
Proxy, ICAP Request Header Insertion	insertheader.req.server.java
Proxy, ICAP Request Header Inspection / Alteration	browseragentchanger.req.server.java
Proxy, ICAP Request URL Conditional Change	pdfonlineviewer.req.server.java
Proxy, ICAP Request to Response Modification	blacklistcheck.req.server.java
Proxy, ICAP Response Content Insertion - Static	test.resp.server.java
Proxy, ICAP Response Content Insertion - Database (Non-caching)	globalmessagesnocache.resp.server.java
Proxy, ICAP Response Content Insertion - Database (Caching)	globalmessages.resp.server.java
Proxy, ICAP Response Content Antivirus Scanning	antivirus.resp.server.java
Proxy, ICAP Request & Response Multiple Operations	insertheader.req.server.java browseragentchanger.req.server.java pdfonlineviewer.req.server.java blacklistcheck.req.server.java test.resp.server.java antivirus.resp.server.java antivirus.resp.server.java

Table 4.4: The test cases used for performance testing

Note: All scripts had modifications undertaken so they worked at full speed and did not leave the local network. For example, pdfonlineviewer redirected to an identical file on the same webserver.

The classes used as control variables were used to ensure that testing functioned to evaluate the system developed, and not a bottleneck elsewhere. These results needed throughputs adjusted when link bandwidth was saturated, as overheads arise with the use of the system that do not arise when it is not present. For instance, the use of a proxy will halve throughput as content must be sent over the network twice, and using an ICAP server will further diminish this to no more than one third of the original bandwidth, as it requires yet another transmission. If the ICAP server modifies something this diminishes again as the content must be sent across the network four times (web server to proxy to ICAP server to proxy to client) This is discussed in the results.

The request and response script classes are comprised of scripts which fundamentally operate upon the body or headers of requests sent by the client or responses sent from the web server. This is how the performance of individual aspects of the system developed is evaluated.

The multiple operations class is used to gain an overview of how performance is affected when multiple operations are used in at the same time. Because Greasyspoon runs the scripts in a serial fashion, this should be approximately the sum of the component performances with overheads removed. Database scripts were not used in this test for two reasons: putting them in parallel would provide a non-realistic use case, and they had major performance issues making the performance tests fail completely.

Load Testing Variables

The load testing variables were used to ascertain how the system performed under various load conditions. The maximum values chosen were used because they hit link limits in many cases, or commonly resulted in system timeout when servicing requests. The variables used are given in table 4.5, and were tested in an automated way (by a Linux shell script) under every possible combination for each script class.

Test duration had a value of 60 seconds. Because the central aim of this project was not performance, this was kept low enough to keep the duration of overall testing manageable. With 1200 tests testing still took 20 hours, if this was set higher, which it probably should be for stability reasons, then testing would take too long. This is mentioned as a possible testing enhancement in section 7.

Variable	Values Used
Test Duration	60 Seconds
Total Number of Requests Made	10 000 (166 / second) 100 000 (1666 / second)
Concurrent Connections	1 10 100 200 400
File Size Used	1KB 10KB 100KB 1MB 10MB
File Contents	Random Binary Textual HTML

Table 4.5: The performance testing variables used to measure system characteristics under load

Total number of requests made, with values of 10 000 and 100 000, was used to provide pure request load conditions that were medium-low, and medium-high respectively.

Concurrent connections, with values of 1, 10, 100, 200, and 400, had the values chosen to range from the trivial control case of 1, to around the point where the system failed to resolve many tests (400).

File size used, of 1KB, 10KB, 100KB, 1MB, and 10MB was designed to spread around a usual webpage size of 100KB in a logarithmic fashion.

File content type, of binary and textual, was used to ascertain if any content compression or string handling facilities were impacting performance.

Chapter 5

Results

While this project generated a large amount of code and experimental data, it is presented here in digest form. We first discuss the completion state of the system developed as it related to the project's required functionality and system requirements. Next the results of testing the effectiveness of the automated mitigation measures against drive by downloads, malicious websites, and virus downloads is discussed. Finally we cover the performance characteristics of the system under various load situations.

5.1 Functionality of System Developed

All functionality was implemented to at least a proof of concept level, undertaken with a view to meeting the overall system requirements given in section 3.4. We shall discuss the implementation state of each functionality class, before moving on to discuss system requirements in the following section. Some of the Java code used is given in Appendix H for background.

5.1.1 User Messaging

All user messaging functionality was implemented to the point where it provided reliable message insertion in HTML based web content. An overview of the status of each specific user messaging functionality is given in table 5.1.

Global Userbase Messaging is the most trivial of the user messaging functionalities, as it only requires inserting a message into the HTML stream and no checks are performed to ascertain anything about the user. This script uses the database to store the messages, and caches the message between regular updates.

Functionality	Status	Notes
Global Userbase Messaging	Implemented	
Specific User Messaging	Implemented	User Selection Mechanism RADIUS or IP Address
Userbase Subset Messaging	Implemented	Arbitrary User Selection Using Database

Table 5.1: User messaging functionality status

Specific User Messaging is an extension upon the global user messaging script where (in addition to checking the message is HTML) the response is checked for user identity before the message displayed is chosen. The user's identity is ascertained from either the proxy's authentication header, or the IP address of the request (as configured to be sent by the proxy). Upon review it was noticed that this identity is put directly in a static SQL lookup, thus leaving the script open up to abuse via SQL injection if a malicious username is used, this should be investigated further and mitigation measures implemented.

Userbase Subset Messaging is an extension beyond messaging a specific user which is accomplished via an additional database join that adds information about the user upon which message selection is based. For example, by joining a table containing user addresses then all users in a specific city can be messaged. As with the user specific messaging, this script is potentially vulnerable to abuse via SQL injection and this should also be investigated further.

5.1.2 Inline Warnings

Inline warnings were implemented, but not to the level originally intended due to difficulties in detection and delivery. An overview of the current status of the inline warning functionality is given in table 5.2 below, and discussion follows. We briefly discuss the issues here, however much of the investigation is relegated to future work.

Functionality	Status	Notes
Browser Version Warnings	Partially Implemented	Sends message via user messaging functionality. User-agent based detection only.
Plugin Version Warning	Partially Implemented	Sends message via user messaging functionality. User-agent based detection only.

Table 5.2: Inline warning functionality status

Detection of some software versions is simple because it is carried in the user agent header sent in requests, however version detection in other client side software would require the injec-

tion of various client side scripts (such as Javascript or VBscript) into the page, and this could cause major compatibility issues. The result of using only user-agent detection is that it is not possible to determine the versions of much client side software beyond the web browser. The injection of version detection scripts should be investigated, but current behaviour of checking every request may need to be changed, perhaps through the use of a single regular check, or a separate page that is used to detect this.

The inline notification is delivered through user messaging. While this does reach the customer, it will go to all devices the customer is using, and not just the device that triggered the warning, potentially causing confusion. Determination of a specific device would require some method of full state handling so that each request can be linked to its own response, and the notification inserted in only that response. While it is possible to currently perform simple linking of request and responses by measures such as page URL or client IP address these can be easily confused in common usage scenarios such as multiple clients browsing the same sites from behind a NAT router. In an organization with tightly controlled address allocation of one per device this could be accomplished, but further investigation is needed.

5.1.3 Automatic Threat Mitigation

Several different methods of automatic threat mitigation by content modification were implemented, and they used varous combinations of header inspection, request redirection, and content scanning to accomplish their tasks. All scripts were implemented fully, however some performance issues remain to be resolved with the more complicated functions. Browser exploitation and drive-by-download mitigation is not a single functionality, but is an effect that is desired of the end system. Discussion of this drive by download mitigation continues in section 5.3.4. The status of automatic mitigation measures is given in table 5.9.

Objectionable Text Warning has been implemented for a static list of words. If anything on the list is seen in the page content (as detected by a regular expression) then the page displayed is replaced with a warning page that the user can bypass. Currently the bypass mechanism is simple, and prone to cross site request forgery (which could be used by attackers to trick browsers into automatically overriding warnings without user input.) should the attacker know the nonce used to bypass. While the execution of the bypass mechanism could be improved (as discussed in section 7.2.1), the functionality works to an adequate level for a proof of concept system.

Functionality	Status	Notes
Objectionable Text Warning	Implemented	Functions via static list in script. Would benefit from flatfile or database use.
Dangerous Filetypes	Implemented	Some dangerous document filetypes redirected to an online viewer sucessfully.
Virus Downloads	Implemented	Antivirus scanning implemented on all downloaded files smaller than 10MB.
Site Blacklist Checking	Partial	Blacklisting via Static list, and database lookup (Google Safe Browsing). Protects against Phishing/Site Impersonation, and known malware sites, but significant performance issues remain
Reflective Cross Site Scripting	Implemented	Simple mitigations remove reflected scripting content successfully
Browser Exploitation mitigation	Partial	More a behaviour than a specific functionality, see effectiveness testing (section 5.3.4) .

Table 5.3: Status of automatic threat mitigation functionalities

Dangerous Filetype Redirection has been implemented for Word 97 (.doc), Word 2007 (.docx), Powerpoint (.ppt and .pptx), and PDF (.pdf) document formats. These filetypes are redirected to an online viewer As drive-by-download mitigation measure, with the intention of isolating the user from any malicious content contained in them. The user can bypass by clicking the download link on the page, and this should be somewhat safe from cross site request forgery attacks as the verification is referrer rather than nonce based. The script used for PDF redirection is given in Appendix H.2.3, while a script that simply blocks PDF files based upon the file header is given in Appendix H.2.2.

Download Virus Scanning, also a drive by download mitigation, is working well for scanning and blocking viruses. The performance impact seems to be less than was anticipated, and the script is given in Appendix H.2.1.

Site Blacklist Checking functions well for static blacklists, however the use of the jgoogle-safebrowsing library resulted in severe performance impacts and concurrency crashes that resulted in it stopping the entire system.

Reflective Cross Site Scripting Warning is functional and effective at blocking simple attacks that utilize the greater than (“<”) character or an encoded variant of it. Although many attacks use this character, it is not a required aspect of a reflected cross site scripting attack. Thus this functionality is adequate for a proof of conecpt model, but a more comprehensive system should be developed for production usage.

Now that the results of implementing specific functionality have been implemented, we will discuss the results of the system as a whole as pertains to the original requirements specified in section 3.4.

5.2 Requirement Evaluation

Five sets of system requirements were given in section 3.4, with some derived from the stated requirements for the Comcast system, and others added to allow the additional functionality developed in this project. These requirements were met in varying degrees; in this section we discuss the state of those requirements' completion. As with the functionalities, we give summaries in tabular form, followed by discussion.

Requirements are designated as either *met* (completed to the level necessary for a proof of concept system), *partially met* (completed, but no way to evaluate completion, or has issues remaining), *not met* (not undertaken or results unsatisfactory for proof of concept), or *abandoned* (later determined to be a non-core requirement). Some of the requirements, which we denote *dependent* were met indirectly and automatically through the characteristics of the software used. Requirements met through this dependency shall only be discussed briefly, as they were trivial to meet.

5.2.1 General Requirements

The majority of the general system requirements were met: out of the ten specified eight were met, one not met, and one abandoned. Of those that were met, three (TCP port 80, active session handling, and non-use of TCP resets) were dependent and met purely by virtue of the technologies used. The overall summary of the status of the general requirements is given in table 5.4.

The block listing requirement (that sites can be exempted from modification) is met by the capabilities of the system, however it has not been deployed in any scripts in use. This behaviour can be in scripts directly via either the @include and @exclude directives for greasyspoon (discussed in section 3.9.3), or by implementing an additional conditional (*if*) statement in individual scripts.

The instant messaging requirement was abandoned as a requirement, as this was not felt to be necessary for an environment in which a secure web gateway might be deployed, such as

Req #	Name	Status	Notes
1	TCP Port 80	Met	Dependent
2	Block Listing	Met	Not Used
3	Instant Messaging	Abandoned	
4	Handling of Active Sessions	Met	Dependent
5	No TCP Resets	Met	Dependent
6	Non-Disruptive	Met	Subjective
7	Notification Acknowledgment	Not Met	Planned
9	Unexpected Content Handling	Met	Leaves Unmodified
10	No Caching	Met	Caching Disabled

Table 5.4: Results concerning system general requirements

a company. A controlled environment will have (ideally) have control over the software used internally, and could thus use instant messaging software that neither interferes, nor is interfered with by, the use of the system. Furthermore, should unauthorized software be used then it is unlikely to be considered a problem if it performs strangely.

The non-disruptive requirement is subjective to assess, as what one user considers useful another may consider disruptive. However, in the many test cases we ran manually on commonly used websites, no unexpected interference considered disruptive was encountered.

The notification acknowledgment requirement was not met and remained unimplemented for two main reasons. Firstly, in a secure web gateway acknowledgment that dismisses a message from future appearances is only acceptable in some cases. If a user downloads a virus, they still should be alerted of future incidents. Secondly, as with user warning bypass, we were unable to develop a satisfactory mechanism for message acknowledgment and dismissal which was not vulnerable to cross site request forgery attacks which could be used by attackers to trick browsers into automatically overriding warnings without user input. We discuss this in section 7.2.1

The unexpected content handling requirement was met in all scripts used. If unexpected content was used then modification was skipped. While as a fail open security mechanism this may lead to weaknesses, web content is composed of such a large number of different content types and scenarios that a pure whitelisting approach will lead to problems for users.

The non caching requirement was met by configuring squid to not cache content.

5.2.2 Proxy Requirements

The proxy requirements (table 5.5) were all met through the use of the same Squid web proxy as used in [20].

Req #	Name	Status	Notes
12	Open Source Software	Met	Dependent
13	ICAP Client	Met	Dependent
14	Access Control	Met	Dependent

Table 5.5: Results concerning proxy server requirements

5.2.3 ICAP Service Requirements

The ICAP service requirements apply primarily to the messaging aspects of the ICAP service, and were all met successfully (see table 5.6). The request and response support was met by the use of Greasyspoon as the ICAP server. The multiple and simultaneous differing message requirements were met in the messaging functionality implemented.

Req #	Name	Status	Notes
15	Request and Response Support	Met	Dependent
17	Multiple Notification [or Modification] Types	Met	Can message arbitrary messages to arbitrary set of users
18	Simultaneous Differing Notifications [or Modifications]	Met	Can message arbitrary different messages to an arbitrary set of different users

Table 5.6: Results concerning ICAP service requirements

5.2.4 Messaging Requirements

The messaging service developed met one requirement and partially met another while two of the requirements were not met. The requirements necessary for messaging operation were met, while those necessary for organizational reasons were not. This is summarized in table 5.7.

The requirement for a (Caching) Messaging Service was partially met. The messaging functionality was successfully implemented, however issues remain to be resolved with the caching of messages from the database. These issues are related to those that occurred with blacklist update for the blacklist functionality , and the resolution of these is issues is relegated to future work and discussed in section 7

Req #	Name	Status	Notes
19	Messaging Service (caching)	Partially Met	Messaging and caching works, some concurrency issues on refresh from database
20	Process Acknowledgments	Not Met	Planned
21	Ensure Notification Targeting Accuracy	Met	No false messaging observed
22	Keep Records for Customer Support	Not Met	Planned

Table 5.7: Results concerning messaging service requirements

The requirements to process acknowledgments and keep records for customer support were not implemented as they were not considered core to the protection functionality of a secure web gateway. They are however important for the deployment of such a system, and while implementing simple handling of these may be undertaken through adding database writes to scripts, a useful implementation of this functionality requires the development of a full user interface and work flow to handle the resulting data. This has been designated as future work and is noted in section 7.2.1.

The requirement of ensuring target notification accuracy was met. During development of the functionality no display of a message to an incorrect user was encountered at any stage.

5.2.5 Additional Requirements

The additional requirements that were defined as necessary to make the system developed useful in a real world implementation were all met to some degree. One was met to a large degree of satisfaction, while the remaining were met partially to a level that was considered adequate for some deployments, but perhaps requiring improvement for others. This is summarized in table 5.9.

Req #	Name	Status	Notes
A1	User Override	Partially Met	Not XSRF safe (Future work: 7.2.1
A2	Automated Update	Partially Met	Software updates enabled
A3	Acceptable Delay	Partially Met	Subjective threshold, see 5.4 for testing results
A4	Acceptable Throughput	Partially Met	Subjective threshold. See 5.4 for testing results
A5	Improved Security Against Malware	Met	Protection comparable to other measures. See 5.3.4

Table 5.8: Status of additional automatic threat mitigation functionalities

The user override requirement was met in a simple way, but not in a way that was considered safe from cross site request forgery attacks which could be used by attackers to trick browsers into automatically overriding warnings without user input (relegated to future work in section 7.1).

The automated update requirement was met for some of the software used that relies upon detection software updates, but not for others. The antivirus signatures automatically updated whilst blacklist updating was problematic. This is discussed in future work (section 7.1).

The acceptable delay and acceptable throughput requirements were tested using the methodology discussed in section 4.3. However, as with any performance characteristics these will be considered to meet an *acceptable* level for some situations but not for others. See section 5.4 later in this chapter for full discussion of the results.

The improved malware security requirement was tested using the methodology discussed in section 4.2 and achieved a degree of protection comparable to other measures. Because a comparable level of protection was achieved this was considered to be sufficient to meet the requirement. See section 5.3.4 later in this chapter for full discussion of the results.

5.3 Effectiveness Testing

Testing the effectiveness of the automated measures at mitigating web threats was undertaken by visiting approximately 90 000 total web pages on the client honeypots, producing results that indicate that protection is comparable to other measures. However, due to unresolved issues with the stability and consistency of the testbed used to evaluate effectiveness, while these results appear to show promise the results are indicative only, and cannot be considered conclusive.

Nevertheless, the results appear to indicate that there may be promise in the full development and implementation of a system upon the foundation of this project. We did find malware, much of it with low detection rates for antivirus. For example, one particular file (named mldkse.exe) which was downloaded via a drive by download from site accessed via a shorturl (bit.ly) link had was detected by 15 of the 41 antivirus programs the VirusTotal [177] service uses to scan submitted files.

The results cannot be considered conclusive for the following reasons: the system under test is at a proof of concept stage and is not considered production ready, the sample size of the tests and number of test runs is insufficient to perform an adequate statistical analysis, and some test error cases could not be adequately managed.

Seven total test runs were performed: three test runs that were used to refine the process, two test runs using a set of known malicious URLs, and two using URLs collected at the time of testing. The summary of the test run results is given in table 5.9.

5.3.1 Data and Measurements

Within each of the four capture-hpc client honeypot testbeds each site in the url list was visited, and the results reported by the client component inside the virtual machine were recorded.

Capture-hpc categories

Each URL visited by Capture-hpc was recorded as either *safe* (no errors and no malicious activity detected), *malicious* (No errors and unauthorized activity detected), or *urlError* (errors loading page).

The preliminary runs had error rates that were near complete for the system developed, and differed widely between the other protection mechanisms. This caused problems as unless these rates were at a similar level then any effectiveness test results were unreliable and invalid. The error rates were improved by reducing host system load, and solving or isolating the causes of

Run #	Type	Status	Notes
1	Preliminary (Mixed Type)	Preliminary Run	ICAP System failure after 4 minutes due to concurrency error in database update
2	Preliminary (Mixed Type)	Preliminary Run	Complete System Crash in Unprotected System (Resource overallocation). ICAP system restarted numerous times
3	Preliminary (Mixed Type)	Preliminary Run	Database timeouts in ICAP system, resource overallocation led to numerous VM timeouts on all systems
4	Collected URLs	Indicative	Safesquid performed twice as badly as unprotected (Result questionable). System developed stopped about one third compared to the unprotected system while antivirus performed stopped around half.
5.1	Known Bad	Indicative	Antivirus performed well, safesquid marginally better than unprotected and system developed marginally better than safesquid.
5.2	Known Bad	Indicative	Continuation of run 5.1 after multiple system crash. Antivirus performed well, safesquid marginally better than unprotected and system developed marginally better than safesquid.
6	Collected URLs	Indicative	Antivirus performed only slightly better than unprotected, antivirus and safesquid each prevented around two thirds of that detected in unprotected system.
7	Known Bad	Test Failure	Safesquid system had no results, furthermore the error and malicious levels on other systems varied from previous tests. (Results Discarded)

Table 5.9: Summary of effectiveness test results by run

major performance and reliability issues on the developed system.

For each malicious site a record was sent to the capture server before reverting the VM to a clean state. The record sent to and stored by the server contains a list of the modifications detected and a copy of the files modified, added or deleted. Although significant effort was undertaken to improve the whitelist which defined safe events, a substantial number of false positive malicious results were encountered and later manually screened out. A heavily simplified sample set of executions extracted from the record of a drive-by-download encountered is given in Appendix I.

URL Error rates

Early test runs had page error rates that differed widely between the different detection mechanisms. This indicated that the results could not be relied upon to indicate anything useful. The error rates were improved in later tests by reducing the load upon the honeypot host systems, and by removing problematic scripts from the test. The error rates for the three test runs are

shown in figure 5.1.

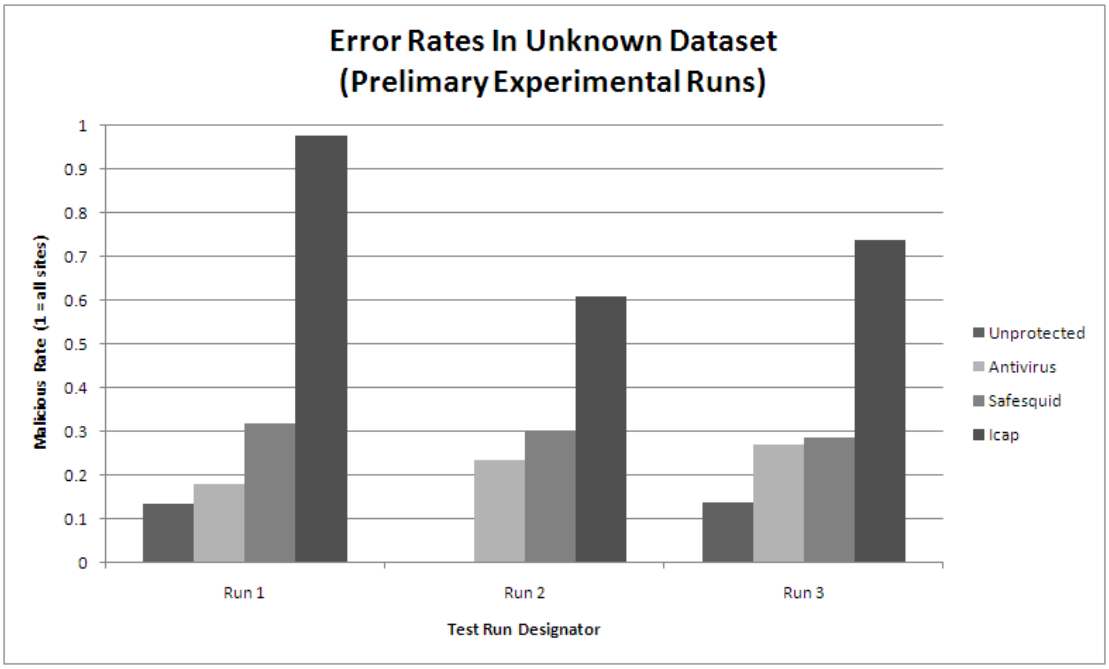

Figure 5.1: Error rates in preliminary runs

Host machine load was reduced by decreasing the number of concurrent virtual machines hosted. Reducing the number of concurrent virtual machines from four in the first run, to three in the second improved the rate marginally, in the third run logging was set to a higher level so that the source of the problem could be found. Reviewing Greasyspoon logs after the third run indicated that problems were caused by scripts that block while waiting for a database request or update. The Google Safe Browsing blacklist check script was disabled as it was blocking on update and also encountering errors in spikes in the number of concurrent connections (which occur when the honeypot visits a batch of sites).

After another unrecorded short test it was observed that the error rates had converged to approximately similar levels, and for further tests the error rates within a test class were comparable. The error rates for the experimental runs over known URLs are given in table 5.2, and while not completely consistent they were close enough for indicative testing. Run 6 had lower rates compared with run 4 due to optimizations undertaken to in the testbed network and router.

Furthermore, error rates in the known bad data set were consistent also, as shown in figure 5.3. They were much higher, however this is was expected, as any known malicious site can have a very short lifetime and be taken down or moved well within the three day window.

With more consistent error rates, it was possible to place some degree of confidence in the

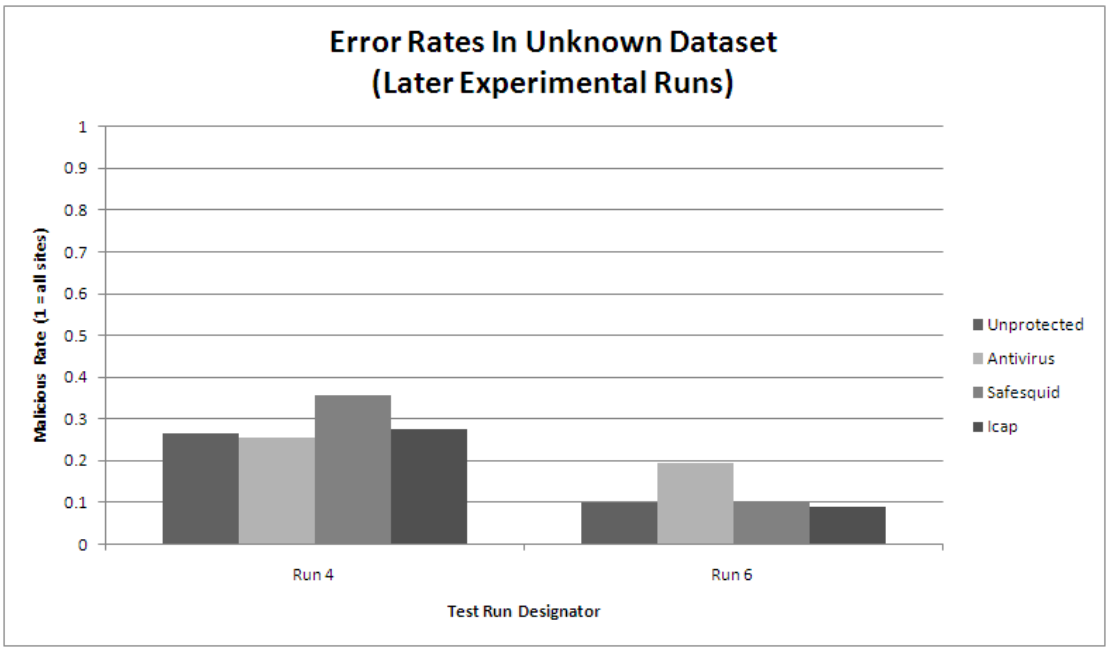

Figure 5.2: Error rates in later experimental runs

indicative nature of the results. However, false positives for malicious sites had to be screened out of the data.

False Malicious Results

All positive results from testing were manually inspected and compared with known or expected operations within the virtual machine, this resulted in the vast majority of the malicious results being recategorized as false positives[1].

False positive were detected at a much higher rate in the known bad set (figure 5.4) than in the unknown set (fig 5.5). Furthermore in the known malicious set the antivirus exhibited a far higher rate of false positive than the other measures, but this trend did not carry over to the collected URLs.

The false positives were caused a combination of the additional software installed and technical problems including: monitoring problems with the Capture client (including whitelist omissions), unanticipated non-malicious network connections, and web servers running on unusual ports.

[1]Any cases of a crashed web browser were categorized as malicious, as they may have been a case of a failed memory corruption attack, such as a buffer overflow

Figure 5.3: Error rates in known bad experimental runs

Monitoring problems with the capture client were the vast majority of the false positive results. In many of these it appeared that the capture client had not correctly hooked file operations, as the logged operations indicated that a process (normally Internet Explorer's iexplore.exe) had modified the file "C:", which is not a valid windows file path[2]. This indicates that the Capture file monitor was misreading or not hooking the filesystem properly. Reinstall of the tools normally fixed this, however it sometimes seemed to occur spontaneously. These problems were reduced in later tests through improving the whitelists based upon the results of the test runs and decreasing system load reducing timeouts and monitoring errors.

Unanticipated non-malicious network connections caused the second largest group of false positives. Verifying the status of these involved checking the connection header (banner grabbing) where possible, as some servers had since gone offline in the hours since testing (these were categorized as malicious as most genuine web servers are not that transient). The vast majority of the unanticipated connections made were undertaken by client software running on the virtual machine to undertake operations that had not been whitelisted. These operations

[2]A valid Windows file path needs at least another "\" to indicate a directory, or a filename as well to indicate a file. For example "C:\ " and "C:\file.txt" are valid.

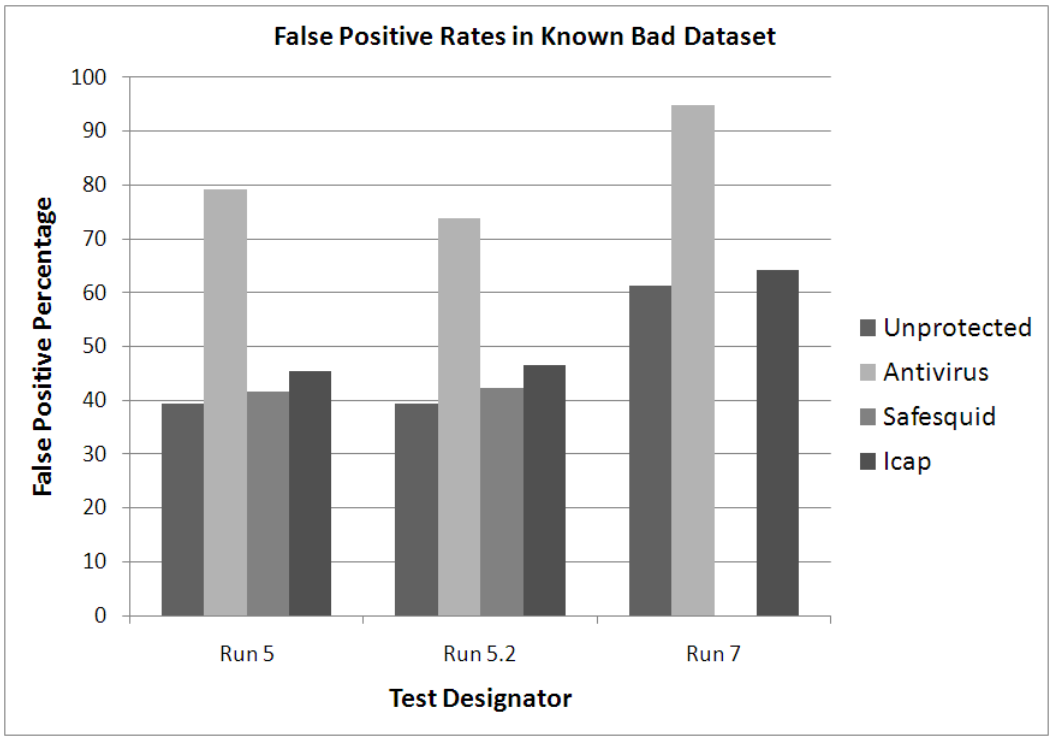

Figure 5.4: False positive rates in known bad experimental runs

included:

- Port 554 - Real Time Streaming Control Protocol (RTSP) through either Quicktime or Windows Media Player
- Port 843 - Flash Socket Policy Server [3]
- Port 1935 - Real Time Messaging Protocol (RTMP), streaming media generally used by the flash player
- Port 6667 - Internet Relay Chat (IRC) - Could be either malicious or non-malicious
- Shoutcast Streaming Media Servers (Various Ports)

Some combinations of these also occurred, for instance a page which used port 843 and 6667 was a flash based IRC client that automatically connected. Also encountered were HTTP servers running on high ports other than 8080 for no apparent reason. Usually running the open source Java based Jetty webserver [78], these were flagged as malicious. This was because it fits with the expected situation for compromised machines as a part of a fast-flux network.

[3] Used to check with a host if flash is allowed to open a TCP connection to it [174]

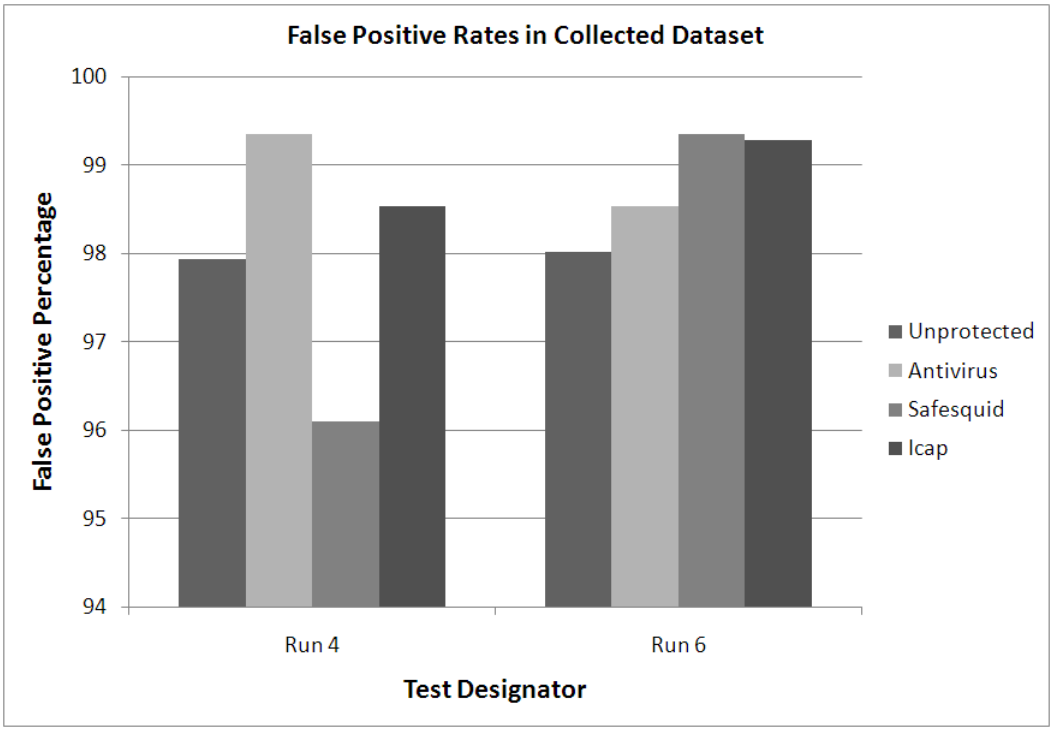

Figure 5.5: False positive rates in unknown/collected runs

Inconsistent behaviours and strange results

There were some issues with websites not acting in a completely predictable manner, with malicious activity occurring sporadically, randomly, or once and no more. This is not an entirely unexpected behavior, as it has been reported in [130, 106, 148]. Two measures were investigated as methods to manage the impact of this upon experimental results, the addition of a referrer header to requests and the use of a larger sample set .

A blank referrer has been reported as a method in use by some exploit kits to detect honeypots which visit the site directly rather than visiting it from a link on another page, and if they suspect that a honeypot is visiting they either do not respond or present non-malicious behaviour[172, 64]. To mitigate this an additional squid proxy was inserted on the gateway in transparent mode and was set to replace the header of blank referrer with a referrer of twitter.com search which the results came from. However, the method used for this, squid Access Control Lists (ACL) will only modify existing headers, and cannot insert completely new headers. This could have been achieved through the use of another ICAP server to handle header insertion, however this would have overly complicated the experiment with minimal additional benefit. If this testing methodology is used in future the handling of referrer headers should be investigated, as discussed in section 7.3

To increase the sample size a larger number of sites was visited than was thought strictly necessary to investigate a proof of concept system's effectiveness. The testing undertaken was probably not sufficient to completely mitigate uncertainties and errors; for instance in run 4 Safesquid got a higher malicious rate than the unprotected system (which should not happen). Future testing should use a much larger sample size to deal with this better as we discuss in 7.3

5.3.2 URLs Used

The URLs used in effectiveness testing were in two sets: known malicious (or known bad), and gathered (or unknown).

The known bad URLs were taken from the malwareurl free database, a three day delayed blacklist. The lists were initially in Comma Separated Value (CSV) format in chronological order with other data including threat categories (such as phishing, fraud, or malware). Immediately before testing the latest copy of this list was obtained and the URLs for the top one thousand malware related results were taken and used for the test run.

The Automatically Gathered URLs were gathered by searching Twitter messages for links in messages mentioning topics that were popular on Google Search and Twitter messages. Sometimes the names of prominent people feature highly, while some trends are are cryptic or quirky. Experimental run 4 was run around the time of the 2010 soccer world cup final and some of the trends that were used to gather URLs are shown below. Note that the names of many soccer players appear.

- Andy Schleck
- Barefoot Bandit
- Bob Sheppard
- Closing Ceremony
- colton harris moore
- dani jarque
- dumb and dumber
- Dunga Burro
- espana campeon
- Espanha
- Felipe Melo
- FIFA World Cup
- free slurpee day 2010
- Furia Roja
- Hannah Montana Forever
- Holanda
- Hup Holland Hup
- Iker Casillas
- iniesta shirt
- jimmy buffett
- jimmy buffett tour dates 2010
- kevinhart
- lala vazquez
- margaritaville
- Museumplein
- Nelson Mandela
- Netherland
- Ocho Cinco
- Omaha Mall
- one love one ocean
- paula creamer
- Pulpo Paul
- shamicka gibbs
- Sneijder

- spain wins world cup
- tna victory road
- uncle henry s
- Vuvuzela
- Walter Hawkins
- Webber
- wewasgooduntil
- who won the world cup 2010
- whyyouatchurch
- worldcup

Using these trends, the links that were found in the at the same time in messages were gathered. Over the experimental runs that used gathered URLs, every URL used by capture directly was from one of around around 5000 domains. Due to the source of the links (Twitter) these were primarily the domains of URL shorteners, with the bit.ly service constituting by far the most used service. The distribution of the top ten domains is illustrated in figure 5.6.

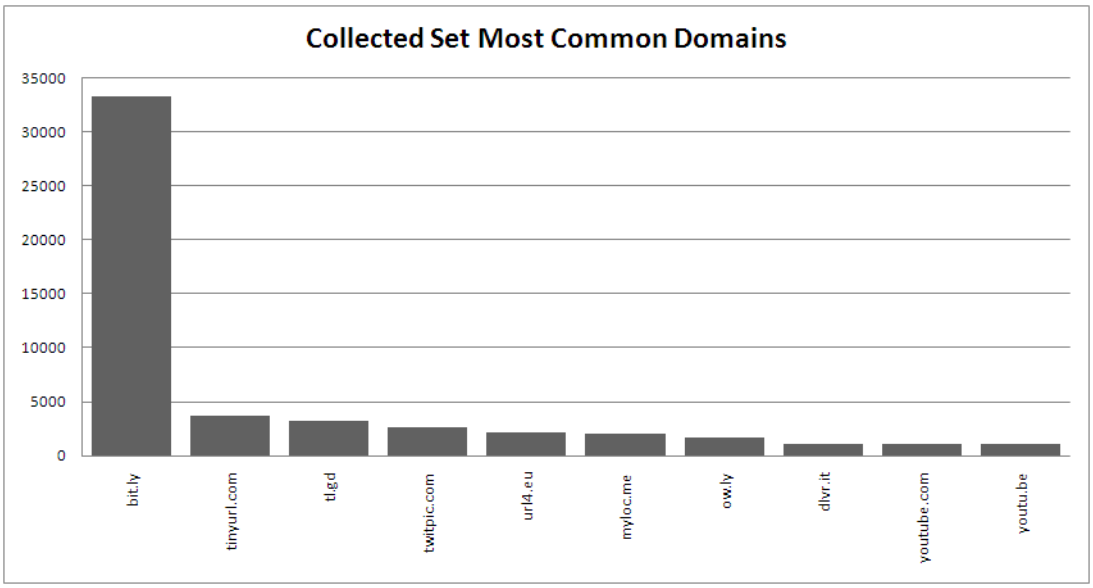

Figure 5.6: The most common domains among gathered internet trends

Although it may initially seem that 5000 domains is a small number for around 100 000 URLs, the use of URL shorteners means that the variety of actual domains crawled is far more diverse.

5.3.3 Effectiveness results in known bad set

For the runs over the known malicious dataset results from the the final run (run 7) are not considered due to the error rate (inconsistent and Safesquid completely non-functional - see fig 5.3) and small size of the test (only one hour of testing).

In the tests that the results are considered for (5.1 and the continuation of it 5.2), the effectiveness of the system developed appears to block some malicious activity compared with the unprotected system, but only around 15% of it. This is a very small result, and while it appears

to indicate some improvement, the antivirus system shows a far high level of mitigation. It is worth noting that if the blacklist had been put into the blocklist for the System developed the mitigation rate would have been 100%.[4]

This is probably for two reasons: antivirus engine detection levels and the location of the antivirus system in the dataflow. Detection levels are likely to be high in AVG in this case because a delayed (public) blacklist has given the antivirus provider three days to develop and deploy signatures or blocklists. ClamAV also has virus signatures, but also a well known poor detection rate. The use of a more effective virus detection engine may significantly improve this for our system.

The position of antivirus in the dataflow may also lead to a higher detection rate irrespective of the engine used. This is due to the fact that an antivirus engine running on the client machine can inspect data right before it is executed and as it is executed on the system. Many techniques used to evade detection (such as encoding, signature evasion, droppers, or loader packing) will tend to be unravelled as the malware is run on the target system.

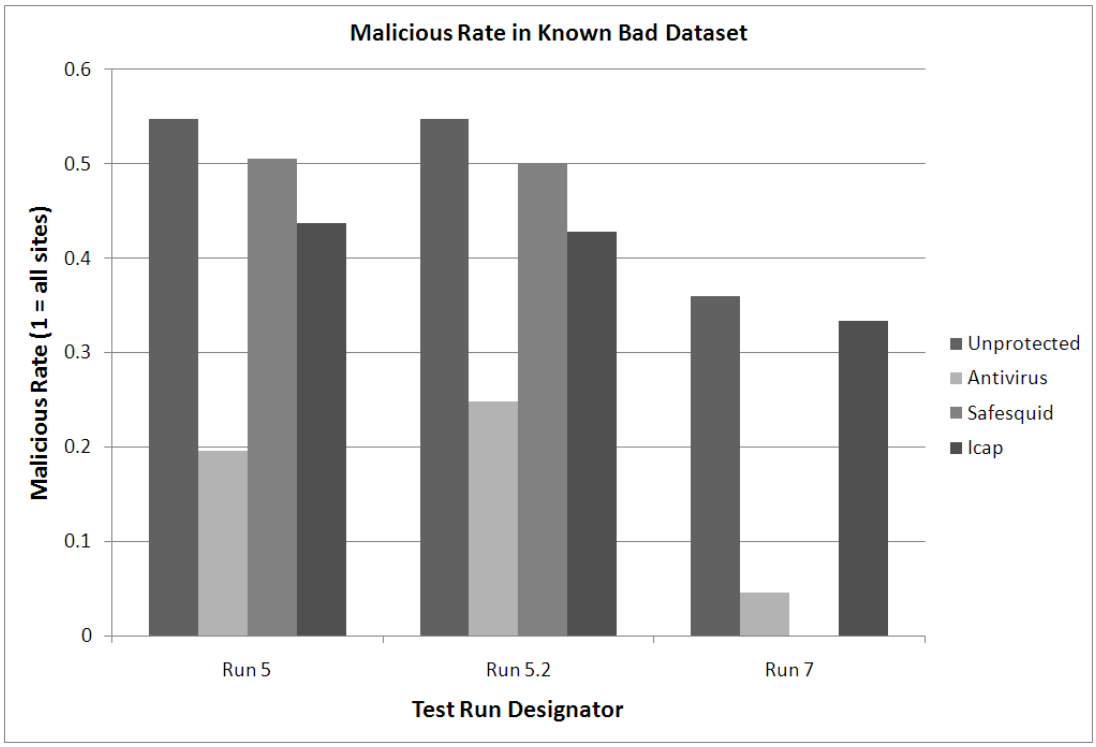

Figure 5.7: Malicious rates in experimental runs over known bad URLs

It is also suspected that the weight of the entries on the list used towards windows executable files (.exe) may have caused some of the difference in rate. This is because many of the measures

[4]Only in this case however as we would be blocking our test data. If we had used one blacklist to test another against we would have had a rate in the middle representing the overlapping entries, no blacklist is perfect.

we deploy are good for use against drive by downloads that exploit weaknesses in software that uses other data formats to deploy their payload. This is far more prevalent in the testing of the URLs collected from trends.

5.3.4 Effectiveness Results in Unknown Set

The rates of malicious sites detected in this set, while consistent with expectations from the literature at around half of one percent (as discussed in section 2.7.3), have the effect of adding a large degree of uncertainty to the results due to the effective sample size. Testing was only undertaken for indicative results and not conclusive results however, so while this should be considered when drawing conclusions from these results it does not lead us to entirely discard them.

Effectiveness

The verified malicious rates detected for the collected URLs were significantly lower than the rates in the known malicious URLs. Where detected malicious rates in the the known malicious were around one in four (25%), for the collected set it was one in two hundred (0.5%), or fifty times lower. The two runs used for data collection, four and six, present substantially different relative protection measurements for the protection measures. Run six approximately doubles the malicious rate for unprotected and Safesquid, triples it for antivirus, but the level for the ICAP system remains roughly the same. This is probably due to the large difference in the number of pages visited between the protection measures within each run affecting the sample used. This is shown in fig 5.8, which gives the number of sites visited.

When the results of the two runs are averaged, the new sample sizes range from around 5000 to around 11000. Spread remains between sample sizes, but there is a larger and more similar sample set overall[5]. The detected malicious results from the two have also been averaged (weighted by number of pages visited) to show the value across the larger sample set. The two experimental runs and their weighted averaged results are shown in fig 5.9.

Run 4 has two areas of interest that indicate possible problems in the results: the Safesquid system shows a malicious rate that is substantially higher than the unprotected system (which is a control) in run four, and the antivirus system shows a higher weighted average than the unprotected system.

[5]Averaging these values, and the results below is not entirely statistically valid as the results are not entirely independent – there are some URLs that are in both sets

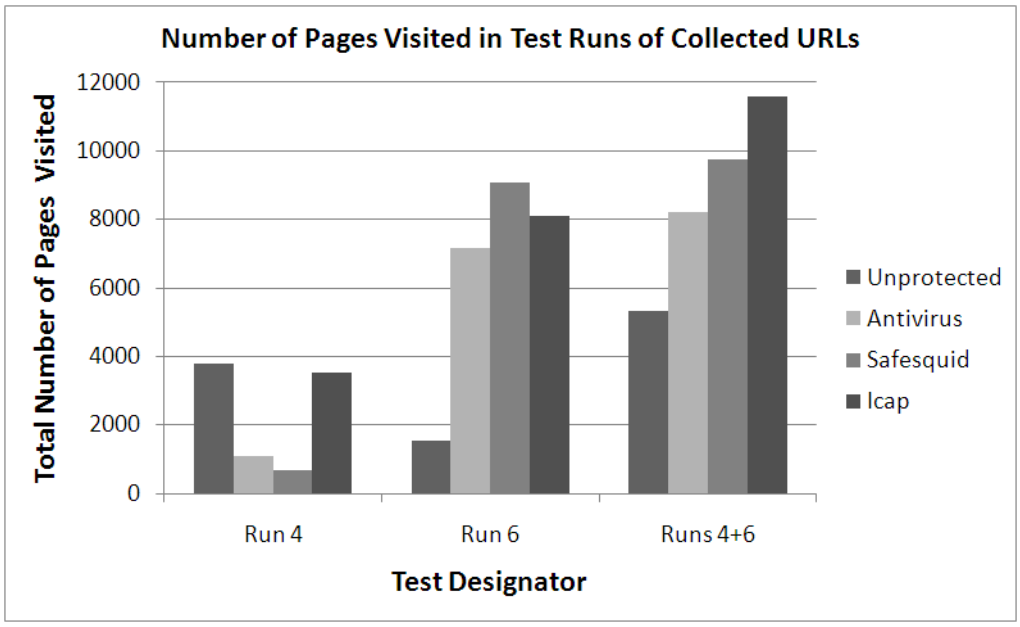

Figure 5.8: Number of URLs tested in experimental runs over collected URLs

The Safesquid result in run four is likely due to the small sample size used in that run (687), which is far smaller than the number of URLs visited by the other measures For a sample set that size there are around 8 positive results, and a one or two positive results will alter the rate by a large amount.

The cause of the antivirus having a higher malicious rate than unprotected in the weighted results is less clear. The antivirus outperforms the unprotected in both runs, but when weighting for the number of requests is undertaken the result is a higher rate. The rate of extra positive results needs to be three times higher in run six to account for the rise, which in the same size used (8000 sites) would be a substantial swing. There are two likely contributers to this outcome: undetected false positives in the antivirus result set and errors in testing methodology.

The antivirus system exhibited far more false positives during testing across the known malicious URLs than in the unknown set. It is possible, that false positives were not detected and removed from the data as effectively in this data set. Examining the data regarding crashing and non-crashing and non crashing malicious results hints that this may be the case. Figure 5.10 shows the raw breakdown for malicious results in run six (not adjusted for sample size), and it is notable that the ratios between crashing and non-crashing malicious results are quite different between the two proxy based systems and the unprotected and control systems.

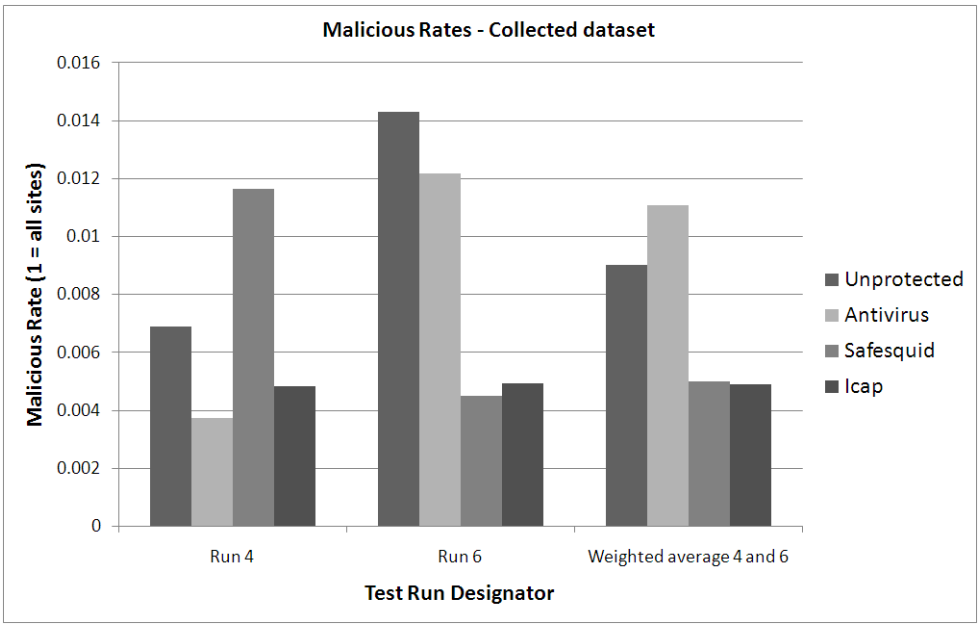

Figure 5.9: Malicious rates in experimental runs over collected URLs

Figure 5.10: Run six malicious results by web browser crash status

It is suspected that a bad whitelist combined with an incomplete knowledge of the system processes and modifications made by the AVG antivirus resulted in many normal acitivites that AVG undertakes being classified as malicious. Additionally, the much larger number of false positives given in the known malicious data sets contribute to this conclusion.

Errors in testing may also have led to the unexpected results, as while the honeypots were given the same URL list and the same duration to run, they did not sample exactly the same sites,

or experience the same behaviours. This is due to different relative crawl speeds, and also due to the non-casual behaviour of some malicious websites. An improved method for undertaking this style of comparative protection effectiveness testing using client honeypots that should improve this is proposed in future work section 7.3.2.

5.4 Performance Testing

Twelve representative script classes were tested through one hundred combinations of load that affected performance. The tests varied the filesize of the requested file, number of requests made per second, number of concurrent requests made, and file makeup (either random binary or textual). All tests had a duration of 60 seconds and while this is quite a short time (system stability issues may take longer to appear), the duration was kept low to make the number of data points and total testing duration more manageable. The test cases for each script class were manually compiled into tabular form (as shown by the table of results from a single test case in Appendix I.2), from which the results discussed here were further processed.

5.4.1 General Performance response

The variables altered during testing either caused performance effects proportional to their additional network load, or increased throughput marginally until an inflection point was encountered after while the performance gains reversed. The results for the four independent variables used are given in table 5.10.

Variable	General effect of increase	Notes
File Contents	Slightly faster for binary transfer than HTML in most cases.	Results cannot be due to compresssion, as random data is far less compressible than the HTML data used.
File size	Increased data throughput rate, increased response time	Little additional effect until network link saturation
Request Rate	Little effect	Little impact until resource saturation when this causes drastic decrease in performance
Concurrency	Increased throughput rate and decreased response time	Provides benefits until link or component resource saturation

Table 5.10: The general results of performance testing as they relate to each independent variable used

5.4.2 File Contents

Request modification results are not relevant to this case, as the difference is in the response contents. The two forms of file content used, random binary and HTML, were not expected to exhibit much performance difference except in cases where content scanning would not modify the file (due to it not being able to be parsed as a string or not containing some data being checked for). However in testing a consistent performance advantage was measured, in both control cases and non-control cases. The improvement was affected by connection concurrency, but a pattern was unable to be discerned as concurrency did not appear to have a consistent effect. This is shown in figure 5.11.

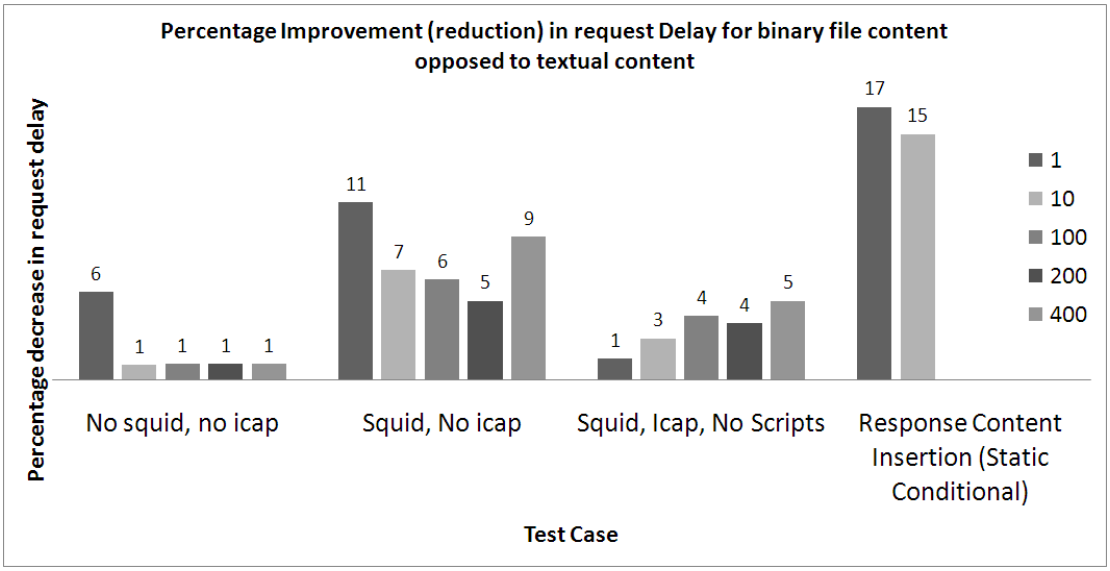

Figure 5.11: Percentage decreases in request duration from binary content files over HTML content files (10 000 requests / minute, 100KB filesize, grouped by connection concurrency)

The difference seen in the final case on the graph, around 15%, is due to a combination of the normal improvement seen in the control cases and reduced processing and network overhead that results from the binary file not requiring modification as the textual string searched for "*<body ... >*" is not contained in the file.

5.4.3 File Size

File size showed performance characteristics similar to what would be expected. Smaller files have a larger relative overhead (due to network connection handling and HTTP headers) than larger files, but larger files require more network bandwidth to transfer. Thus, after smaller files it was expected that the general trends would scale with the filesize. This is what was observed,

as is illustrated in figure 5.12, although the 10KB filesize gave slightly shorter times than the 1KB filesize.

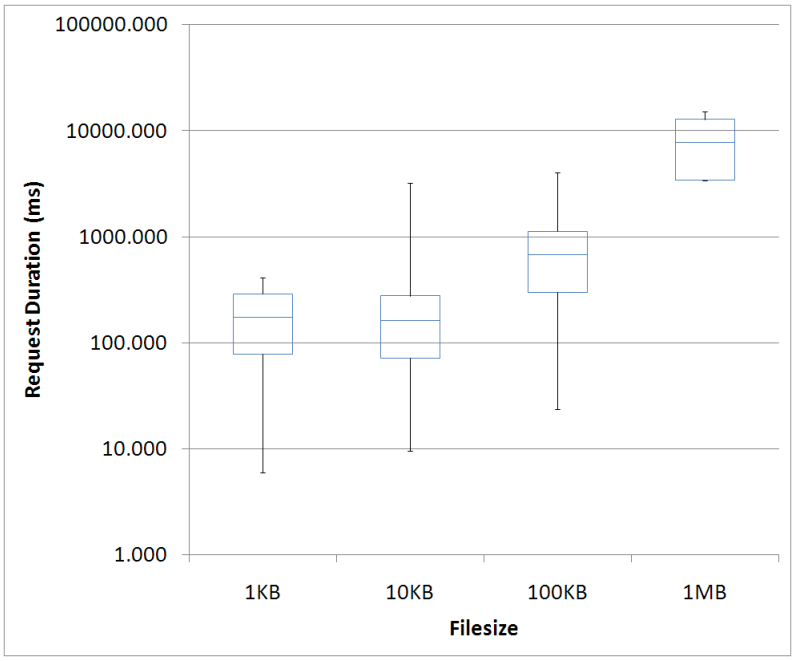

Figure 5.12: Effect of filesize upon time to service request for antivirus. (10000 requests / minute, 100KB filesize)

5.4.4 Request Rate

Request rate did not show much change in experimental results other than increasing duration by a factor of ten and moving timeout failure conditions down in most test cases by an order of ten. This was primarily due to network and TCP connection saturation. For instance, where some tests previously began to timeout with concurrency one-hundred they began to exhibit timeout with concurrency of ten. This particularly problematic with large filesizes as with 1667 requests per second link saturation was reached at a file sizer of around 68KB, and in most cases large error and timeout rates occurred in the 100KB and larger test cases.

5.4.5 Connection Concurrency

Connection concurrency of the values tested provided peak performance (in requests per second) at ten. A concurrency of less than ten appeared to be underutilized, whilst concurrency of two-hundred and four-hundred causes the number of requests serviced per second to drop.

Figure 5.13 shows the response to concurrency exhibited by testing direct communication with the web server. Both logarithmic concurrency sequences (1, 10, 100 and 100, 200, 400)

follow straight lines on the graph with a logartithmic scale and are therefore follow linear patterns of performance. If the concurrency is doubled then the connection delay approximately doubles.

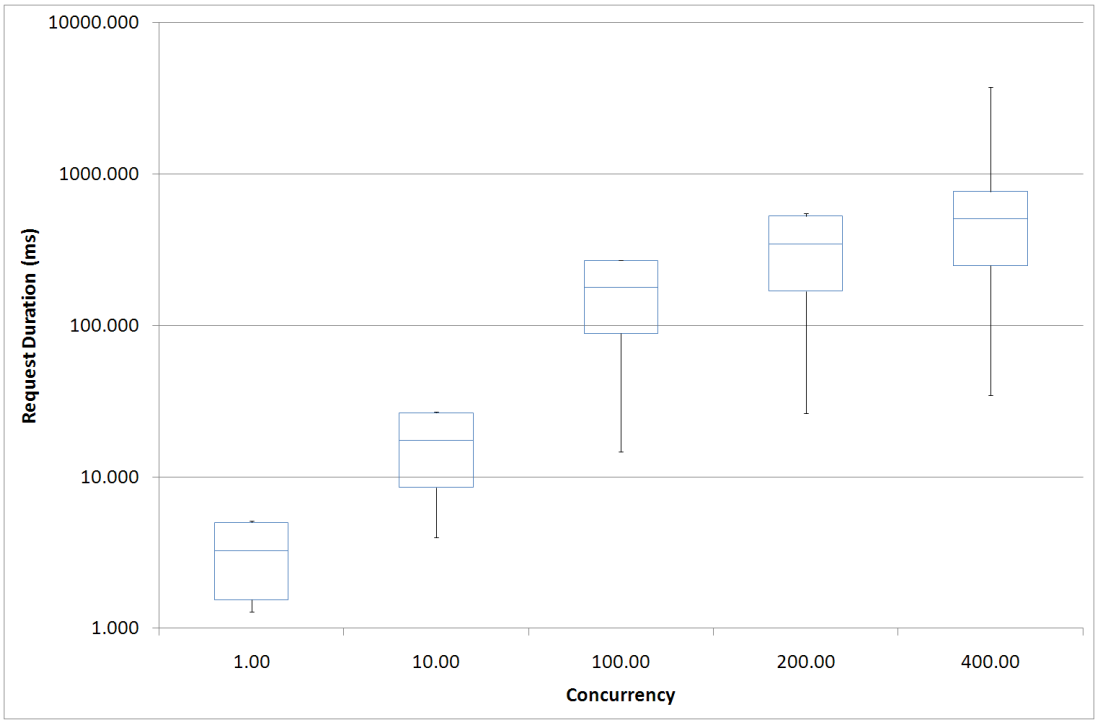

Figure 5.13: Effect of concurrency upon time to service request (Non-proxied, 100 KB filesize, 10 000 requests per minute)

With the ICAP header modification request script the relationship between concurrency and time taken to service requests changes to only increase after ten requests per second. This is illustrated in figure 5.14.

Furthermore, the rate of increase between the later points becomes higher, indicating that diminishing returns occur as concurrency increases. Thus, concurrency adds performance benefits up to a point, but beyond it it actually decreases overall performance.

5.4.6 Performance response with different script types

The different operations performed require a differing degree of network communication and system resources, and they therefore have differing performance characteristics. From the representative cases and controls used in the performance testing the general observations in table 5.11 occurred.

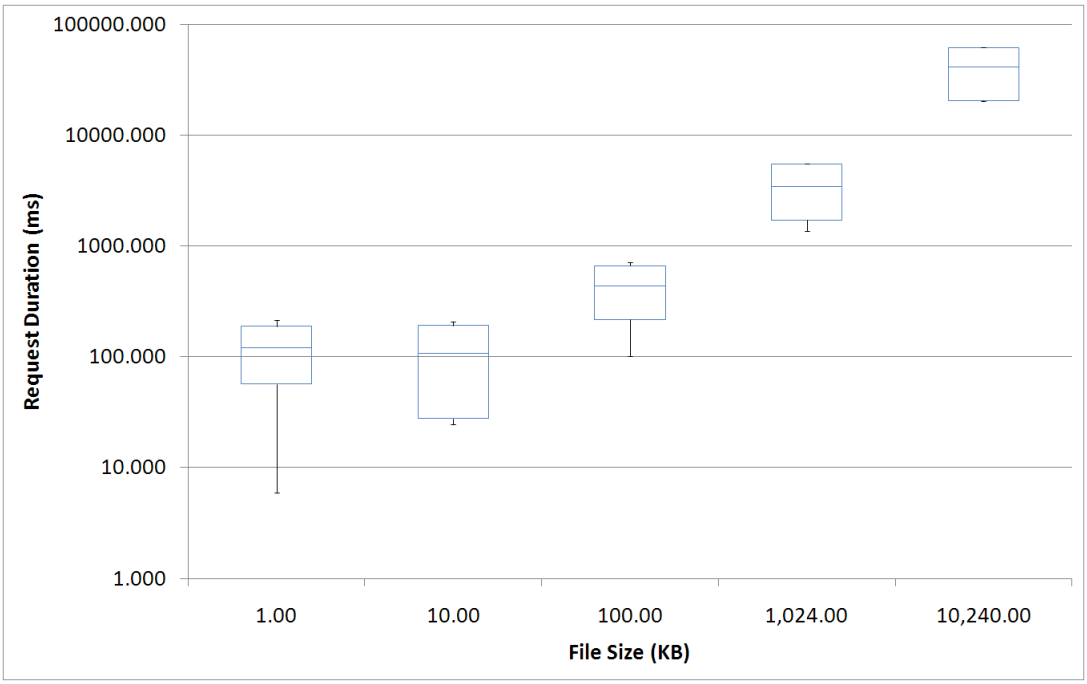

Figure 5.14: Effect of concurrency upon time to service request (ICAP header modification)

Script Class	**Performance Notes**
ICAP Request Header Insertion	Little performance impact
ICAP Request Header Alteration	Little performance impact
ICAP Request URL Change	Little performance impact
ICAP Request to Response Modification	Little performance impact
ICAP Response Content Insertion - Static	Little performance impact, except in high concurrency when timeouts occurred
ICAP Response Content Insertion - Database (Non-caching)	Performance results very low, about 10 seconds per request. Omitted from graphs for clarity
ICAP Response Content Insertion - Database (Caching)	Little performance impact, except on requests that require a database update. Caching mechanism problematic, should be removed to external triggers and not controlled by Greasyspoon.
ICAP Response Content Antivirus Scanning	Moderate performance impact, except in high concurrency when timeouts occurred due to network link saturation .
ICAP Request & Response Multiple Operations	Operations performed serially by Greasyspoon, so delays are cumulative. Moderate performance impact when problematic operations are removed.

Table 5.11: Script class results for performance testing

Request Scripts

The performance of the requests script classes was very good, an overview is shown in figure 5.15. Note that the values have been adjusted to compensate for number of network transfers necessary for each type (for instance direct transfers a file once, proxied twice, ICAP three times).

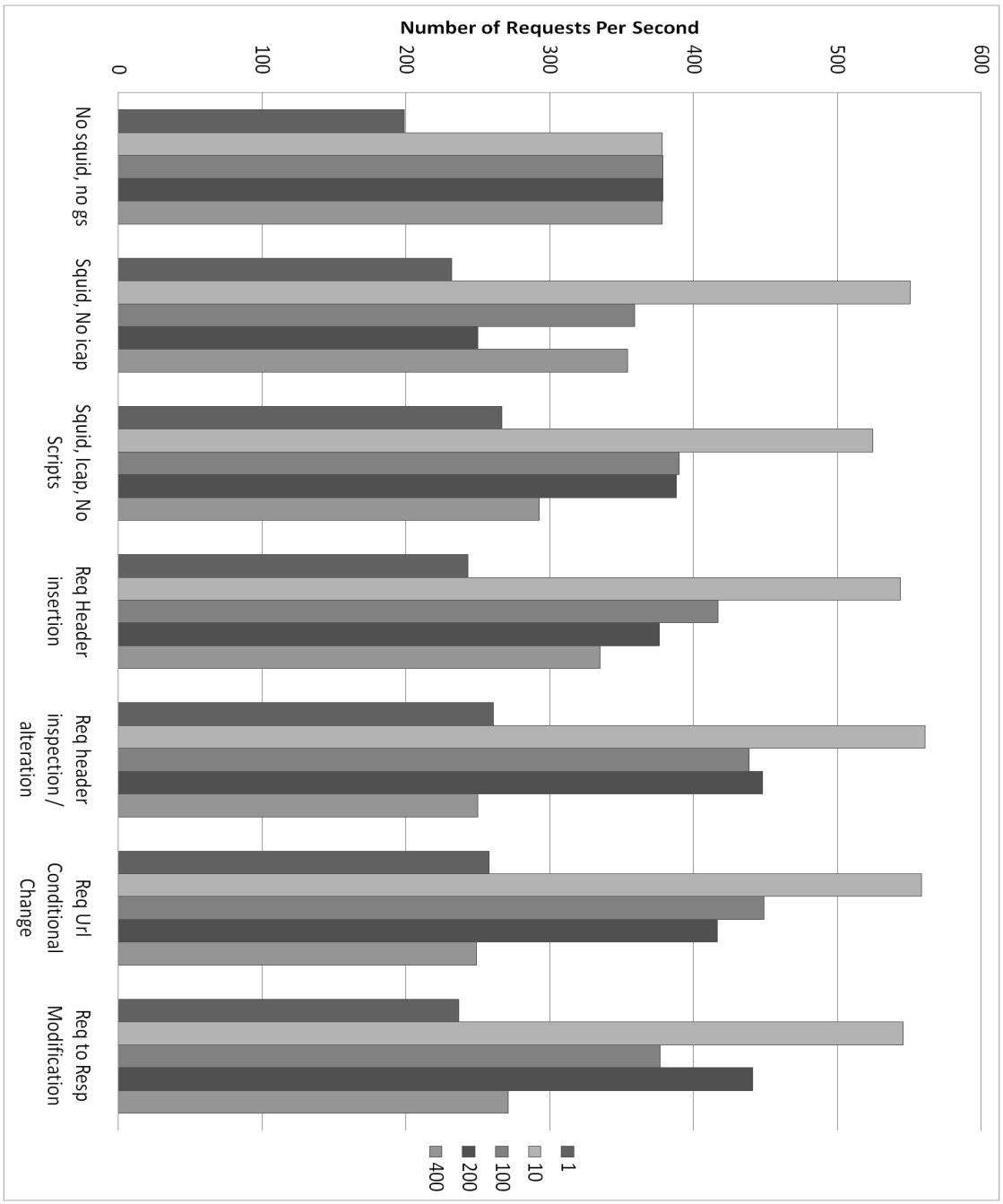

Figure 5.15: Performance comparison for request scripts (10 000 requests per minute, 100KB filesize)

Although worse than the control in all cases except concurrency ten[6], every test case used for the 10 000 requests / minute runs gave a throughput of better than 200 requests per second. While this is not enough for an enterprise grade system, it is more than enough for a home or small business system and for a proof of concept system this indicates adequate performance is possible for request operations..

Response Scripts

Response operations had a much larger performance overhead than the request scripts; primarily this was due to the larger file sizes and more complex operations performed. Where a request is a few kilobytes at most, the test cases shown in the figures below used 100KB files, and in cases of higher concurrency this appears to have caused performance problems.

Note also that the values for both figures 5.16 and 5.17 have been adjusted to compensate for number of network transfers as in the requests case above. From figure 5.16 it may be observed that the database accessing content insertion script had performance issues (already discussed), and that the response scripts developed had issues beyond 100 requests per second.

[6]It is suspected, but not verified) that the improvement in concurrency ten is due to the web proxy keeping the TCP connection open.

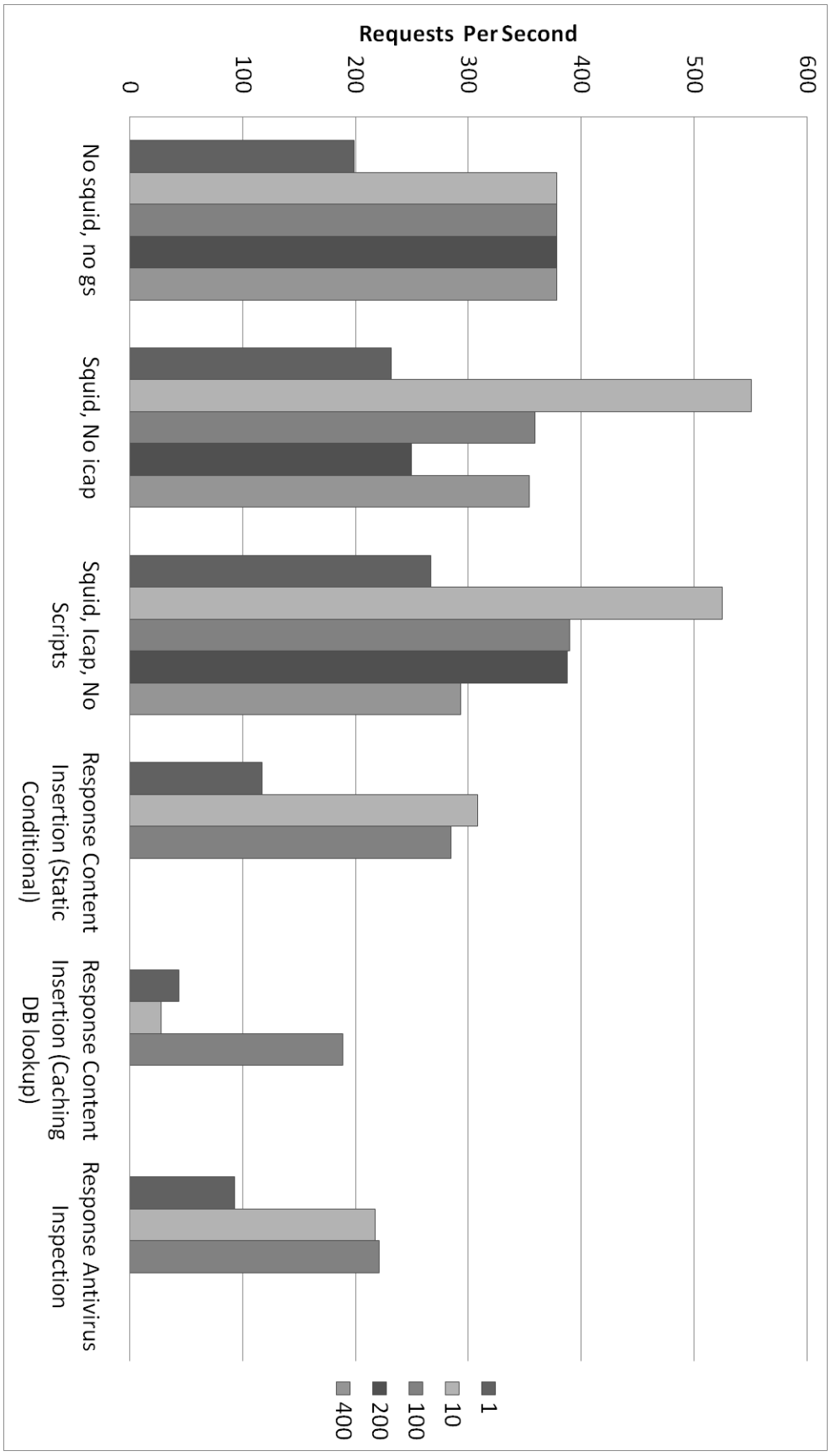

Figure 5.16: Requests per second performance for different response script types

To clarify the reasons for this, response times rather than connections per second are used in figure 5.17.

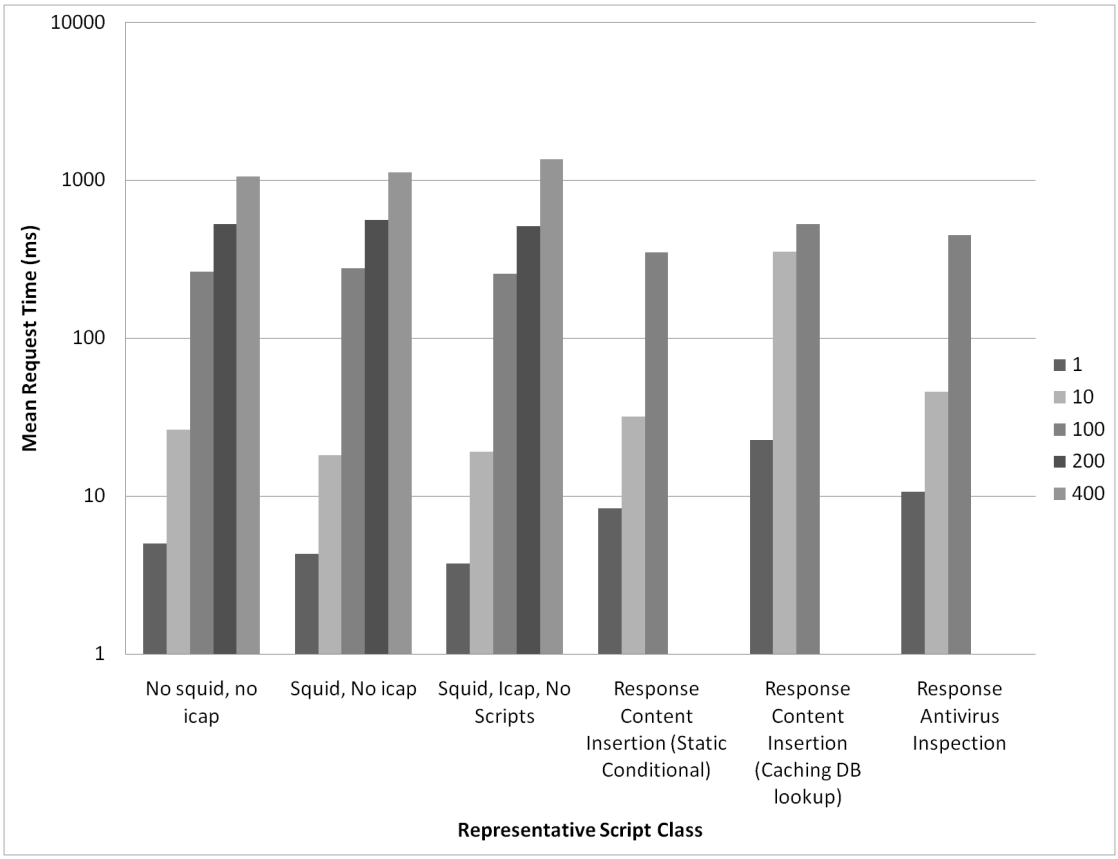

Figure 5.17: Response times for different script types (10 000 requests / minute, 100KB file size, grouped by concurrency)

The response scripts add little to response time when concurrency is low, but at high concurrency the errors that occur cause connection timeout.

5.4.7 Performance results

The performance characteristics as currently implemented can handle request operations adequately for a home or small business, but response operations can be problematic. For deployment the following are thought to be the maximum reasonable capacity: Eighty-percent network link load (after all multiple transmission are accounted for), concurrency of twenty, file size of less than one megabyte (standard web browsing), and a request load of no more than ten users. Much of this would be significantly improved if all components were deployed on the same physical system, thus network throughput is far less significant an issue.

Chapter 6

Conclusions

6.1 Project Summary

This work describes the development, deployment, and testing of a secure web gateway using open source tools. The system developed shows promise for meeting the goals which were set for it to a proof of concept level. Testing showed some improvement in browser infection rates, but performance problems with the system developed combined with stability problems in testing mean that, while the results appear promising, conclusive results as regarding a full deployment should not be drawn until system performance and testing reliability are improved. Due to the complexity of the project, full conclusions shall be broken down into four areas: An overall review, conclusions regarding system effectiveness at threat mitigation, conclusions regarding system performance, and conclusions regarding the project hypotheses.

6.1.1 Development Review

The messaging service met requirements outlined to a proof of concept level. It met all requirements outlined with the exception of the requirements concerned with customer accounting. This functionality is non-complex to implement and is unlikely to have had any effect upon the results of the testing phases of the project.

The additional requirements necessary for the active security measures were all met to a proof of concept level where they were well-defined. The requirements for “acceptable” performance were less clearly defined, nevertheless the authors are of the opinion that they have been met for the majority of the project’s functionality.

6.1.2 Effectiveness Conclusions

The testing undertaken indicates that improved security was achieved for web clients over an unprotected system. Testing stability was problematic, and should be considered indicative but non-conclusive. Further effectiveness testing should be undertaken after improving system performance and effectiveness testing reliability, but was not undertaken during the course of this project due to time constraints.

The developed system exhibited slightly lower rates of malicious activity in the three-day delayed known malicious dataset than the control and Safesquid systems, but higher rates than the antivirus system (which had far lower malicious rates than the other methods) in this URL set. This is suspected to be due to the antivirus system signatures having at least three days in which to update, and the known poor detection rates of ClamAV which was used as the antivirus engine for Safesquid and the system developed.

In the collected URL dataset, which had an unknown rate of malicious activity and was gathered in real time while testing, the system developed showed malicious activity rates around half those of the control system in the weighted results. When weighted results are used, the developed system performs similar to the Safesquid system, but the Antivirus system shows higher malicious rates than the control, indicating that problems exist in the testing methodology. This is suspected to be due to the interaction between AVG and Capture-HPC causing web browser crashes which were flagged as malicious.

6.1.3 Performance Conclusions

The system performed very well on request modification operations, but performance of the response operations was mixed. Concurrency of ten requests provided the best throughput, but request rate and file size had little impact (beyond that expected by the larger proportional size of overheads in smaller files) when the link was not saturated. The system exhibited slight performance improvements with binary over textual files which is probably due to non-textual files following a shorter code path.

6.1.4 Conclusion Regarding Hypotheses

The primary hypothesis of this work, that ICAP can act as an application layer IPS for web content, was not disproven, and results of this work indicate that it probably holds true.

The first secondary hypothesis (regarding improving web browsing security) appears to hold

when rates are compared to the control, and indicative results show it is comparable in protection to competing measures.

The second secondary hypothesis (regarding performance) appears to hold for request operations in the system developed, but at present does not hold for response operations. This is likely due to factors regarding performance optimization which can be remedied in future work.

Chapter 7

Future Work

This project has a broad range of future work that can be undertaken to further knowledge and expand upon the ideas investigated. These include improving and optimizing the system developed, extending the system developed, improvements and further investigations around the effectiveness testing, improvements and further investigations around the performance testing, and further developing the system developed into a full system usable in production environments.

7.1 Developed System - Fixes and Optimization

Some issues arose during development or testing need further investigation, primarily concentrated around response modification operations. The following should in particular be investigated or improved:

- Proxy caching (also the effect of this upon security operations)
- Deployment of system components on the same physical system
- Concurrency issues in updating static and semi-static content such as blacklists and messages
- Database server and schema optimizations
- The use of a separate web server to handle messaging content (as used by Comcast[21])

7.2 Developed System - Enhancements and extensions

If performance surrounding core functionality is sufficiently improved then some extensions of functionality that may be useful are suggested. These include general functionality improvements and the implementation of specific additional functionality.

7.2.1 General Extensions

Cross Site Request Forgery Safe Bypass Links

At present the bypass links are potentially vulnerable to Cross Site Request Forgery attacks that a sufficiently informed attacker may use to manipulate web browsers into silently bypassing. The implementation of a Cross Site Request Forgery safe warning bypass should be investigated. Suggestions for this include linking requests, responses, and database entries to allow the use of a random user specific, site specific and time limited nonce appended to links. (e.g. **http://www.website.com/file.exe&bypassAV=98FhwmdnSWl** will download file.exe even if the antivirus flags it as infected, but only that file, from that site, for that user within the next ten minutes)

User accounting and support records

Although we have scripts to read the database, and write operations are trivial to implement, scripts that performed database writes were not implemented due to the performance issues encountered surrounding database access. If this is improved then it is likely to be beneficial for scripts to store accounting and user confirmation data in the database.

Encrypted (SSL/HTTPS) Content Modification

The Squid web proxy has two recently introduced features (introduced while this project was in the testing phases) that may allow this system to terminate the encrypted session, inspect the content, and re-encrypt the content to send back to the client. The first feature is (termed SSLBump [141]) enables Squid to terminate and reestablish secure connections, and the second (Dynamic SSL Certificate Generation [140]) enables Squid to sign the client connection so that certificate errors do not occur indicating a man in the middle attack (provided the proxy signs with a root certificate that the client has installed). At present Squid is unable to perform the dynamic certificate generation in transparent mode.

There are many issues surrounding this functionality, and regulatory, privacy, and legal factors must be considered in addition to the technical issues. In corporate or other tightly controlled environments these are far less likely to be an issue and this functionality will allow additional protection, as at present encrypted content cannot be inspected in any capacity.

7.2.2 Specific Improvements

The following improvements should be investigated as improvements to existing scripts:

- **Antivirus** - The use of an antivirus engine with better detection rates (since ClamAV is known to have bad rates [37]), or the use of multiple antivirus engine scanning
- **Antivirus** - File hashing and hash result caching, so that files need not be rescanned multiple times
- **Version Checking** - Check using more advanced methods than the user-agent sent (such as injecting javascript that checks)
- **User Messaging** - Improve targeting to specific browsers and not just IP addresses
- **Cross Site Scripting** - Couple requests and responses to check for more advanced attacks

7.2.3 Partially or Conceptually Developed Functionality

Some functionality was not considered to be at a stage to be tested, and was only implemented in a very rudimentary fashion or had been researched or planned out, but was not programmed. Nevertheless some of this is worth noting here.

- **Anonymizer** - Removes or replaces user specific content from some sites (such as User-agent, IP Address, installed Plugins, Cookies, Flash cookies)
- **Dataloss Prevention (DLP)** - Removes or warns user if certain patterns or information is sent out in requests. This could include company secrets, Credit Card numbers, or personal details.
- **Standard Signature Scanning** - Scans files for signatures in industry standard formats such as that used by the SNORT Network Intrusion Detection System.
- **Short URL Expander** - Intercepts short URL links and replaces them with their full target address
- **Advertisement Blocking** - Blocks known advertising network sites and domains
- **User Setting Control** - Integrates with the database to enable the user to control the security operations performed by the system on their traffic inline (discussed further on)

7.3 Effectiveness Testing

The effectiveness testing undertaken was using a novel methodology, unfortunately this also resulted in issues being encountered that could not be easily resolved, particularly around the areas of whitelisting and interaction between the monitoring components of the honeypot and antivirus. Future work should be undertaken to improve the validity of results drawn from testing in this project, and the testing methodology could be improved to enable more general real world testing of web client security systems.

7.3.1 Improvements to Testing of Developed System

The effectiveness testing undertaken provided lots of data, but the any conclusions that can be drawn are only tentative. To improve this the following recommendations are made before further testing is undertaken:

- The use of a larger sample size and more test runs in both known bad and collected data sources
- Testing the system with honeypots that are utilizing a more representative software set and not deliberately vulnerable.
- Inserting an additional proxy and ICAP server to handle referrers sent to malicious web sites so that honeypot detection is more difficult
- Investigate the use of a honeypot that crawls websites to collect the links rather than working through ba list of pre-provided links

7.3.2 Extend methodology and investigate as a more general security metric

The testing methodology could be used as a new method for evaluating web client security provided that any testing investigates the previously discussed enhancements. There are additional functions that should be investigated, including the development and use of a proxy that records sites on first pass, and then sends this cached identically to all clients in an attempt to prevent the effects of randomness and blacklisting techniques used as honeypot evasion on results.

7.4 Performance Testing

Although the performance testing results showed quite consistent results, they were dependent upon system performance issues which remain to be solved. We recommend that the effects of the following upon performance be investigated:

- The effect of performance optimizations (such as all components deployed on same physical system) upon performance in the current system
- The effect of using an alternative web proxy (other software, or hardware based)
- The effect of deployment in load-balanced situations
- The effect of additional System resources upon performance
- Testing with a broader range of file sizes and types
- Testing with content gathered from real websites in the same way a real web browser would access it (many files, not all accessed at once, but a few at a time in order)
- The use of in-script performance monitoring and adaptive setting change.

List of Figures

1.1 The address bar in Internet Explorer's latest version (9 Beta), with the difference between sites using Extended Validation certificates and those using other types of valid certificates circled. 7
1.2 Personal Antivirus -a fake antivirus that was delivered through malicious ads on the New York Times webpage in 2009[176] 9
1.3 A simplified software hierarchy illustrating the linkages that may exist between web browser plugins and third party software 11
1.4 An example of a shortened link on a social networking site 13

2.1 A proxied HTTP connection with a single proxy 18
2.2 The basic structure of an HTML document 20
2.3 Mozilla's examples on the JavaScript Same Origin Policy [111] 23
2.4 A screenshot of the Mosaic 3 web browser. [9] 24
2.5 A Proxied connection utilizing ICAP 28
2.6 Numbered steps in a proxied connection utilizing ICAP 29
2.7 An example ICAP request (modified from RFC3507[46]) 31
2.8 In an ICAP or OCP based security system, the ICAP Server handles the security operations. 34
2.9 A proprietary web security proxy handles all content sanitation itself 35
2.10 Client Based Web Security proxies through a local proxy 35
2.11 Bit.ly's short URL statistics page 38
2.12 The type II virtual machine architecture, as used in this research (VMware Server) [136] 40
2.13 Chrome's multi-layered sandboxing, from [134] 46
2.14 The Firefox Plugin version check page [110] 47
2.15 The inline warning web page overlay as used in Comcast Comstant Guard from [121] 57

3.1 The Firefox "reported attack site" primary warning 61
3.2 The Firefox "reported attack site" warning bar, which remains visible if the user continues past the warning 62
3.3 A logical architectural overview of System developed 68
3.4 The Greasyspoon architecture – from the website [105] 70
3.5 The Greasyspoon ICAP server basic architecture and operational workflow [105] 71
3.6 A Venn diagram of the proposed feature set and some other solutions 75
3.7 An example insert shown on trademe.co.nz 77
3.8 The insert in non-minimized form 78
3.9 The insert minimized 78

3.10 A remote PDF file being displayed in a web browser inside the Google Docs online viewer . . . 86
3.11 Order of operations for the antivirus script . . . 86

4.1 Our testbed components . . . 91
4.2 Capture-HPC server layout . . . 92

5.1 Error rates in preliminary runs . . . 114
5.2 Error rates in later experimental runs . . . 115
5.3 Error rates in known bad experimental runs . . . 116
5.4 False positive rates in known bad experimental runs . . . 117
5.5 False positive rates in unknown/collected runs . . . 118
5.6 The most common domains among gathered internet trends . . . 120
5.7 Malicious rates in experimental runs over known bad URLs . . . 121
5.8 Number of URLs tested in experimental runs over collected URLs . . . 123
5.9 Malicious rates in experimental runs over collected URLs . . . 124
5.10 Run six malicious results by web browser crash status . . . 124
5.11 Percentage decreases in request duration from binary content files over HTML content files (10 000 requests / minute, 100KB filesize, grouped by connection concurrency) . . . 126
5.12 Effect of filesize upon time to service request for antivirus. (10000 requests / minute, 100KB filesize) . . . 127
5.13 Effect of concurrency upon time to service request (Non-proxied, 100 KB filesize, 10 000 requests per minute) . . . 128
5.14 Effect of concurrency upon time to service request (ICAP header modification) 129
5.15 Performance comparison for request scripts (10 000 requests per minute, 100KB filesize) . . . 130
5.16 Requests per second performance for different response script types . . . 132
5.17 Response times for different script types (10 000 requests / minute, 100KB file size, grouped by concurrency) . . . 133

A.1 Direct connection in simplified HTTP transaction . . . 160

List of Tables

2.1 Internet Explorer Browser and OS/Service Pack Version Compatibility 25
2.2 Browser security feature support (Sept 2010) 48

3.1 Testbed architecture components & network details 68
3.2 The Greasyspoon scripts implemented 76

4.1 The software installed on the client honeypot 93
4.2 The Capture-HPC server settings in config.xml 96
4.3 The automated response scripts used in effectiveness testing 97
4.4 The test cases used for performance testing 100
4.5 The performance testing variables used to measure system characteristics under load 102

5.1 User messaging functionality status 104
5.2 Inline warning functionality status 104
5.3 Status of automatic threat mitigation functionalities 106
5.4 Results concerning system general requirements 108
5.5 Results concerning proxy server requirements 109
5.6 Results concerning ICAP service requirements 109
5.7 Results concerning messaging service requirements 110
5.8 Status of additional automatic threat mitigation functionalities 110
5.9 Summary of effectiveness test results by run 113
5.10 The general results of performance testing as they relate to each independent variable used 125
5.11 Script class results for performance testing 129

A.1 Some relevant media types 167

B.1 Testbed components and network details 169

Bibliography

[1] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06*. Rio de Janeiro, Brazil.: ACM Press, 2006, p. 41.

[2] Adeptol, "Adeptol Document Viewer." [Online]. Available: http://www.ajaxdocumentviewer.com/

[3] Adobe, "Configure Flash Player auto-update notification." [Online]. Available: http://kb2.adobe.com/cps/167/16701594.html

[4] ——, "Disable automatic updates — Acrobat, Reader." [Online]. Available: http://kb2.adobe.com/cps/402/kb402050.html

[5] C. Alme, "Web Browsers : An Emerging Platform Under Attack," 2009. [Online]. Available: http://newsroom.mcafee.com/images/10039/wp_webw_browsers_w_en.pdf

[6] Anti Phishing working Group, "Phishing Activity Trends Report, 3rd Quarter / 2009," Aug. 2009. [Online]. Available: http://www.antiphishing.org/reports/apwg_report_Q3_2009.pdf

[7] Apple, "Apple Introduces Safari for Windows." [Online]. Available: http://www.apple.com/pr/library/2007/06/11safari.html

[8] ——, "Apple Unveils Safari." [Online]. Available: http://www.apple.com/pr/library/2003/jan/07safari.html

[9] N. C. F. S. Applications, "Screenshot of the Mosaic 3 Web Browser." [Online]. Available: http://gladiator.ncsa.illinois.edu/Images/press-images/mosaic.gif

[10] AV-Comparatives, "Anti-Virus Comparative of products not included in our regular tests," 2007. [Online]. Available: http://www.av-comparatives.org/seiten/ergebnisse/2ndgrouptest.pdf

[11] AV-comparatives.org, "Anti-Virus Comparative On-demand Detection of Malicious Software - No. 25 February 2010," 2010.

[12] T. Beauvisage, "Computer usage in daily life," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI '09*. Boston, MA, USA: ACM Press, 2009, p. 575.

[13] T. Berners-Lee, R. Fielding, and H. Frystyk, "RFC1945: Hypertext Transfer ProtocolHTTP/1.0," *RFC Editor United States*, 1996. [Online]. Available: http://www.apps.ietf.org/rfc/rfc1945.html

[14] T. Berners-Lee, "Information Management: A Proposal," 1989. [Online]. Available: http://www.w3.org/History/1989/proposal.html

[15] ——, "Hypertext Markup Language (HTML) - A Representation of Textual Information and MetaInformation for Retrieval and Interchange," 1993. [Online]. Available: http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt

[16] H. Bos, "Shelia: a client-side honeypot for attack detection." [Online]. Available: http://www.cs.vu.nl/~{}herbertb/misc/shelia/

[17] D. Bradbury, "The battle of the internet browsers," *Infosecurity*, vol. 7, no. 2, pp. 34–37, Mar. 2010.

[18] Chromium-authors, "ChromiumBrowserVsGoogleChrome." [Online]. Available: https://code.google.com/p/chromium/wiki/ChromiumBrowserVsGoogleChrome

[19] Chromium-Authors, "Terms and Conditions - Chromium - Google Code." [Online]. Available: https://code.google.com/chromium/terms.html

[20] C. Chung, A. Kasyanov, J. Livingood, N. Mody, and B. V. Lieu, "Example of an ISP Web Notification System - Draft 00," 2009. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-00

[21] ——, "Comcast's Web Notification System Design," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-09

[22] ——, "Example of an ISP Web Notification System - Draft 01," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-01

[23] ——, "Example of an ISP Web Notification System - Draft 02," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-02

[24] ——, "Example of an ISP Web Notification System - Draft 03," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-03

[25] ——, "Example of an ISP Web Notification System - Draft 04," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-04

[26] ——, "Example of an ISP Web Notification System - Draft 05," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-05

[27] ——, "Example of an ISP Web Notification System - Draft 06," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-06

[28] ——, "Example of an ISP Web Notification System - Draft 07," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-07

[29] ——, "Example of an ISP Web Notification System - Draft 08," 2010. [Online]. Available: http://tools.ietf.org/html/draft-livingood-web-notification-08

[30] ClamAV, "Clam AntiVirus." [Online]. Available: http://www.clamav.net/lang/en/

[31] C. Clementson, "Client-side threats and a honeyclient-based defense mechanism, Honeyscout." Ph.D. dissertation, Linköping University, Sweden, 2009.

[32] Cnn.com, "CNN.com - Breaking News." [Online]. Available: http://www.cnn.com/

[33] Comcast, "Comcast Furthers Internet Security Efforts with National Rollout of Constant Guard Bot Detection And Notification," 2010. [Online]. Available: http://www.comcast.com/About/PressRelease/PressReleaseDetail.ashx?PRID=1012

[34] Common Vulnerabilities And Exposures List, "CVE-2010-0886." [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0886

[35] ——, "CVE-2009-0950," 2009. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0950

[36] ——, "CVE-2010-2568," 2010. [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568

[37] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proceedings of the 19th international conference on World wide web - WWW '10*. New York, New York, USA: ACM Press, 2010, p. 281.

[38] D. Danchev, "Scareware pops-up at FoxNews — ZDNet." [Online]. Available: http://www.zdnet.com/blog/security/scareware-pops-up-at-foxnews/3140?p=3140

[39] Dasient Security, "Malware Attacks on Websites : Answering the Why and How," 2009. [Online]. Available: http://www.dasient.com/resources/why_how_malware_attacks.pdf

[40] E. Davids, "ICAP - The Internet content adaptation protocol," in *AUUG 2004 - Who Are You?*, Melbourne, Victoria Australia, 2001, pp. 71–78.

[41] D. DiNucci, "Fragmented Future," *Print July/Aug*, vol. Jul/Aug, no. July/Aug, pp. 32–34, Aug. 1999. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20461103

[42] W. Dormann and J. Rafail, "Securing Your Web Browser," 2008. [Online]. Available: http://www.us-cert.gov/reading_room/securing_browser/

[43] Ecma International, "ECMA-262 ECMAScript Language Specification," 2009. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[44] M. Egele, E. Kirda, and C. Kruegel, "Mitigating drive-by download attacks: Challenges and open problems," *iNetSec 2009Open Research Problems in*, pp. 52–62, 2009. [Online]. Available: http://www.springerlink.com/index/T26142301U230N58.pdf

[45] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda, *Defending Browsers against Drive-by Downloads: Mitigating Heap-Spraying Code Injection Attacks*. Heidelberg: Springer Britain, Aug. 2009, vol. 1, no. 3, pp. 88–106.

[46] J. Elson and A. Cerpa, "RFC3507: Internet Content Adaptation Protocol (ICAP)," *RFC Editor United States*, 2003. [Online]. Available: http://www.apps.ietf.org/rfc/rfc3507.html

[47] C. Evans, "Encouraging More Chromium Security Research," 2010. [Online]. Available: http://blog.chromium.org/2010/01/encouraging-more-chromium-security.html

[48] C. Evans, J. Tinnes, and M. Zalewski, "Chromium Blog: Improving plug-in security," 2010. [Online]. Available: http://blog.chromium.org/2010/06/improving-plug-in-security.html

[49] T. Fan, Y. Sun, and Y. Zhao, "An Eco-Defending Architecture for Network Security Based on Immuninty and Mobile Agents," in *2009 Fifth International Conference on Information Assurance and Security*. Xi'an, China: Ieee, 2009, pp. 451–454.

[50] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "RFC2068: Hypertext Transfer ProtocolHTTP/1.1," *Internet RFCs*, 1997. [Online]. Available: http://www.apps.ietf.org/rfc/rfc2068.html

[51] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC2616: Hypertext Transfer ProtocolHTTP/1.1," 1999. [Online]. Available: http://www.apps.ietf.org/rfc/rfc2616.html

[52] D. Fisher, "TradeMe virus affects thousands," 2010. [Online]. Available: http://www.nzherald.co.nz/nz/news/article.cfm?c_id=1&objectid=10677853

[53] S. Floyd and L. Daigle, "RFC3238: IAB Architectural and Policy Considerations for Open Pluggable Edge Services," *RFC Editor United States*, 2002. [Online]. Available: http://www.apps.ietf.org/rfc/rfc3238.html

[54] B. J. Fogg, C. Soohoo, D. R. Danielson, L. Marable, J. Stanford, and E. R. Tauber, "How do users evaluate the credibility of Web sites?" in *Proceedings of the 2003 conference on Designing for user experiences - DUX '03*. New York, New York, USA: ACM Press, 2003, p. 1.

[55] M. Forte, W. L. de Souza, and A. F. do Prado, *A content classification and filtering server for the internet*. New York, New York, USA: ACM Press, 2006.

[56] S. Frei, T. Duebendorfer, G. Ollmann, and M. May, "Understanding the Web browser threat: Examination of vulnerable online Web browser populations and the "insecurity iceberg"," in *Proceedings of Defcon 16*, Las Vegas, Nevada, 2008. [Online]. Available: http://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-frei-panel.pdf

[57] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Communications of the ACM*, vol. 45, no. 2, 2002.

[58] Google, "Developer's Guide (v1) - Google Safe Browsing API - Google Code." [Online]. Available: http://code.google.com/apis/safebrowsing/developers_guide.html#Canonicalization

[59] ——, "Google Chrome: A New Take on the Browser." [Online]. Available: http://www.google.com/intl/en/press/pressrel/20080902_chrome.html

[60] ——, "Google Docs - Viewer." [Online]. Available: http://docs.google.com/viewer

[61] GreasySpoon, "Java Examples." [Online]. Available: http://greasyspoon.sourceforge.net/sample_java.html

[62] A. Greenberg, "Symantec Scareware Tells Customers To Renew Or Beg For Mercy - The Firewall - Forbes," 2010. [Online]. Available: http://blogs.forbes.com/andygreenberg/2010/10/04/symantec-scareware-tells-customers-to-renew-or-beg-for-mercy/

[63] O. Hallaraker and G. Vigna, "Detecting Malicious JavaScript Code in Mozilla," in *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*. Shanghai, China: Ieee, 2005, pp. 85–94.

[64] Hamish O'Dea, "The Modern Rogue – Malware With a Face," in *Virus Bulletin 2009*, no. April, Geneva, Switzerland, 2009, pp. 200–213.

[65] P. Harmer, P. Williams, G. Gunsch, and G. Lamont, "An artificial immune system architecture for computer security applications," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 252–280, Jun. 2002.

[66] A. Herzberg and A. Jbara, "Security and identification indicators for browsers against spoofing and phishing attacks," *ACM Transactions on Internet Technology*, vol. 8, no. 4, pp. 1–36, Sep. 2008.

[67] R. Hes, P. Komisarczuk, R. Steenson, and C. Seifert, "The Capture-HPC client architecture," 2010. [Online]. Available: http://ecs.victoria.ac.nz/twiki/pub/Main/TechnicalReportSeries/ECSTR09-11.pdf

[68] I. Hickson, "HTML5: A vocabulary and Associated APIs for HTML and XHTML," 2010. [Online]. Available: http://www.w3.org/TR/html5/

[69] Hitslink, "Browser market share," 2010. [Online]. Available: http://marketshare.hitslink.com/browser-market-share.aspx?qprid=2

[70] J.-H. Hoepman and B. Jacobs, "Increased security through open source," *Communications of the ACM*, vol. 50, no. 1, pp. 79–83, Jan. 2007.

[71] T. Holgers, D. Watson, and S. Gribble, "Cutting through the confusion: A measurement study of homograph attacks," in *USENIX Annual Technical Conference*, Boston, MA, USA, 2006, pp. 261–266. [Online]. Available: http://www.usenix.org/events/usenix06/tech/full_papers/holgers/holgers_html/

[72] Honeynet Project, "Capture-HPC Webpage." [Online]. Available: https://projects.honeynet.org/capture-hpc

[73] F. Howard and O. Komili, "Poisoned search results : How hackers have automated search engine poisoning attacks to distribute malware ." 2010. [Online]. Available: http://www.sophos.com/security/technical-papers/sophos-seo-insights.pdf

[74] J. Huebner, "ClickOff." [Online]. Available: http://www.johanneshuebner.com/en/clickoff.shtml

[75] ICAP Forum, "Homepage." [Online]. Available: http://www.icap-forum.org/

[76] ICAP-Forum, "Products," 2007. [Online]. Available: http://www.icap-forum.org/icap?do=products

[77] O. Ismail, M. Etoh, and Y. Kadobayashi, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," in *18th International Conference on Advanced Information Networking and Applications, 2004. AINA 2004.* Fukuoka, Japan: IEEE, 2004, pp. 145–151.

[78] Jetty, "Jetty WebServer." [Online]. Available: http://jetty.codehaus.org/jetty/

[79] R. Jones, "Netperf Homepage." [Online]. Available: http://www.netperf.org/netperf/

[80] P. Judge, "Barrauda Labs - Annual Report 2009," 2010. [Online]. Available: http://www.barracudalabs.com/downloads/BarracudaLabs2009AnnualReport-FINAL.pdf

[81] J. Kang and D. Lee, "Advanced White List Approach for Preventing Access to Phishing Sites," in *2007 International Conference on Convergence Information Technology (ICCIT 2007)*. Gyeongju, Korea: IEEE, Nov. 2007, pp. 491–496.

[82] S. Keats, "Mapping the Mal Web , Revisited," 2008. [Online]. Available: http://www.siteadvisor.ca/studies/map_malweb_jun2008.pdf

[83] ——, "Mapping the Mal Web, Revisited," *McAfee SiteAdvisor*, pp. 1–26, 2009. [Online]. Available: http://www.siteadvisor.ca/studies/map_malweb_jun2008.pdf

[84] P. Kijewski, C. Overes, and R. Spoor, "The HoneySpider Network - fighting client-side threats," pp. 1–15, 2008. [Online]. Available: http://www.honeyspider.net/wp-content/uploads/2009/06/hsn-first2008-article-v02.pdf

[85] J. Kim, P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross, "Immune system approaches to intrusion detection a review," *Natural Computing*, vol. 6, no. 4, pp. 413–466, 2007.

[86] B. Krishnamurphy, J. C. Mogul, and D. M. Kristol, "Key differences between HTTP/1.0 and HTTP/1.1," *Proceedings of the eighth international conference on World Wide Web*, vol. 31, no. 11, p. 1737, 1999.

[87] P. Kumaraguru, L. F. Cranor, and L. Mather, "Anti-Phishing Landing Page: Turning a 404 into a Teachable Moment for End Users," in *Sixth Conference on Email and Anti-Spam*, Mountain View, CA, USA, 2009. [Online]. Available: http://ceas.cc/2009/papers/ceas2009-paper-37.pdf

[88] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham, "School of phish," in *Proceedings of the 5th Symposium on Usable Privacy and Security - SOUPS '09*. New York, New York, USA: ACM Press, 2009, p. 1.

[89] G. Lawton, "Open source security: opportunity or oxymoron?" *Computer*, vol. 35, no. 3, pp. 18–21, Mar. 2002.

[90] J. Livingood, N. Mody, M. O'reirdan, and Comcast, "Recommendations for the Remediation of Bots in ISP Networks," pp. 1–30, 2010. [Online]. Available: http://tools.ietf.org/html/draft-oreirdan-mody-bot-remediation-10

[91] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, "BLADE: Slashing the Invisible Channel of Drive-by Download Malware," in *Recent Advances in Intrusion Detection*, no. 1. Springer, 2009, pp. 350–352.

[92] ——, *BLADE*. New York, New York, USA: ACM Press, 2010.

[93] M86 Security, "Web Exploits : Theres an App for That." [Online]. Available: http://www.m86security.com/documents/pdfs/security_labs/m86_web_exploits_report.pdf

[94] ——, "Security Labs Report - January - June 2010 Recap," 2010. [Online]. Available: http://www.m86security.com/documents/pdfs/security_labs/m86_security%_labs_report_1H2010.pdf

[95] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious URLs," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*. New York, New York, USA: ACM Press, 2009, p. 1245.

[96] McAfee, "WhitePages squashes ad networks after finding malware —." [Online]. Available: http://www.mxlogic.com/securitynews/web-security/whitepages-squashes-ad-networks-after-finding-malware515.cfm

[97] ——, "The Webs Most Dangerous Search Terms," 2009. [Online]. Available: http://us.mcafee.com/en-us/local/docs/most_dangerous_searchterm_us.pdf

[98] S. McCarron and M. Ishikawa, "XHTML Basic 1.1," 2008. [Online]. Available: http://www.w3.org/TR/xhtml-basic/

[99] Microsoft, "Microsoft Security Bulletin MS04-028: Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)," 2004. [Online]. Available: http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx

[100] ——, "Microsoft Security Bulletin MS06-001: Vulnerability in Graphics Rendering Engine Could Allow Remote Code Execution (912919)," 2006. [Online]. Available: http://www.microsoft.com/technet/security/bulletin/ms06-001.mspx

[101] ——, "Microsoft Security Intelligence Report - Vol 7 (Jan - June 2009)," vol. 7, no. June, 2009.

[102] ——, "Battling Botnets for Control of Computers Microsoft — Security Intelligence Report," *Microsoft Security Intelligence Report*, vol. 9, no. June, 2010.

[103] ——, "Microsoft Security Intelligence Report Volume 9 (January through June 2010)," vol. 9, no. June, 2010.

[104] Microsoft Security, "Microsoft Security Intelligence Report Volume 8 (July through December 2009) Key Findings Summary," *Intelligence*, vol. 8, no. December, pp. 1–20, 2009.

[105] K. Mittig, "GreasySpoon - home," 2010. [Online]. Available: http://greasyspoon.sourceforge.net/

[106] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy, "Spyproxy: Execution-based detection of malicious web content," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. USENIX Association, 2007, pp. 1–16.

[107] ——, "Spyproxy: Execution-based detection of malicious web content," *USENIX07*. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Spyproxy:+Execution-based+detection+of+malicious+web+content#0

[108] A. Moshchuk, T. Bragin, S. Gribble, and H. Levy, "A Crawler-based Study of Spyware on the Web," in *Proceedings of the 2006 Network and Distributed System Security Symposium*. Citeseer, 2006, pp. 17–33. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2921&rep=rep1&type=pdf

[109] Mozilla, "Electrolysis - MozillaWiki." [Online]. Available: https://wiki.mozilla.org/Electrolysis

[110] ——, "Firefox Plugin Check." [Online]. Available: http://www.mozilla.com/en-US/plugincheck/

[111] ——, "Same origin policy for JavaScript - MDC." [Online]. Available: https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

[112] ——, "Mozilla Security Bug Bounty Program," 2010. [Online]. Available: http://www.mozilla.org/security/bug-bounty.html

[113] MySQL, "MySQL." [Online]. Available: http://www.mysql.com/

[114] Mywot.com, "Safe Browsing Tool — WOT Web of Trust." [Online]. Available: https://www.mywot.com/

[115] J. Nazario, "Phoneyc: A virtual client honeypot," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*. USENIX Association, 2009, p. 6.

[116] New York Times Websit, "Note to Readers - NYTimes.com," 2009. [Online]. Available: https://www.nytimes.com/2009/09/13/business/media/13note.html?_r=2&hp

[117] J. Oberheide and E. Cooke, "Rethinking antivirus: Executable analysis in the network cloud," *Proceedings of the 2nd USENIX workshop on Hot topics in security*, pp. 289–300, Dec. 2007.

[118] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *Proceedings of the 17th conference on Security symposium*. USENIX Association, 2008, pp. 91–106.

[119] T. Oda, G. Wurster, P. van Oorschot, and A. Somayaji, "SOMA: Mutual approval for included content in web pages," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 89–98.

[120] Opera, "Opera: Opera version history." [Online]. Available: http://www.opera.com/docs/history/

[121] J. Opperman, "Security Scene: Constant Guard Rolls Out Nationally : Comcast Voices — The Official Comcast Blog," 2010. [Online]. Available: http://blog.comcast.com/2010/09/security-scene-constant-guard-rolls-out.html

[122] D. S. Peterson, M. Bishop, and R. Pandey, "A Flexible Containment Mechanism for Executing Untrusted Code," *USENIX Security Symposium*, p. 207, 2002.

[123] M. Polychronakis and N. Provos, "Ghost turns zombie: Exploring the life cycle of web-based malware," *In First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[124] A. R. Pop, "DEP / ASLR Implementation Progress in Popular Third-party Windows Applications Table of Contents," pp. 1–9, 2010. [Online]. Available: http://secunia.com/gfx/pdf/DEP_ASLR_2010_paper.pdf

[125] P. Porras, H. Saidi, and V. Yegneswaran, "An Analysis of the iKee. B iPhone Botnet," *Security and Privacy in Mobile Information and Communication Systems*, pp. 141–152, 2010.

[126] G. PORTOKALIDIS and H. BOS, "SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots," *Computer Networks*, vol. 51, no. 5, pp. 1256–1274, 2007.

[127] K. Poulsen, "Cybercrooks Trick Gawker Into Serving Malware-Laced Ad," 2009. [Online]. Available: http://www.wired.com/threatlevel/2009/10/gawker/

[128] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All your iframes point to us," in *Proceedings of the 17th conference USENIX Security symposium*. USENIX Association, 2008, pp. 1–15.

[129] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The ghost in the browser analysis of web-based malware," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, 2007, p. 4.

[130] M. T. Qassrawi and H. Zhang, "Using Honeyclients to Detect Malicious Websites," in *2010 2nd International Conference on E-business and Information System Security*. Ieee, May 2010, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5473642

[131] D. Ragget, A. Le Hors, and I. Jacobs, "HTML 4.01 Specification," 1999. [Online]. Available: http://www.w3.org/TR/REC-html40/

[132] G. Ras, N. G, P. Firstbrook, and L. Orans, "Gartner Magic Quadrant for Secure Web Gateway," 2010.

[133] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3, pp. 461–491, 2004.

[134] C. Reis, A. Barth, and C. Pizano, "Browser security: lessons from Google Chrome," *Communications of the ACM*, vol. 52, no. 8, pp. 45–49, 2009.

[135] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir, "BrowserShield: Vulnerability-driven filtering of Dynamic HTML," *ACM Transactions on the Web*, vol. 1, no. 3, pp. 11–es, Sep. 2007.

[136] A. R. Reserved, "Virtualizing I/O Devices on VMwareWorkstations Hosted Virtual Machine Monitor," in *Proc Usenix Annual Technical Conference*, vol. 7, Mar. 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4812171

[137] I. Ristic, "Internet SSL Survey 2010," *Proc. Black Hat USA 2010*, vol. 2010, 2010. [Online]. Available: https://media.blackhat.com/bh-us-10/presentations/Ristic/BlackHat-USA-2010-Ristic-Qualys-SSL-Survey-HTTP-Rating-Guide-slides.pdf

[138] D. Roman, "The Mobile Road Ahead," *Communications of the ACM*, vol. 53, no. 10, pp. 10–10, 2010.

[139] A. Rousskov, "RFC 4037: Open Pluggable Edge Services (OPES) Callout Protocol (OCP) Core," *IETF, March*, pp. 1–57, 2005.

[140] ——, "Features-DynamicSslCert," 2010. [Online]. Available: http://wiki.squid-cache.org/Features/DynamicSslCert

[141] A. Rousskov and A. Balabohin, "SslBump," 2010. [Online]. Available: http://wiki.squid-cache.org/Features/SslBump

[142] Rsnake, "XSS (Cross Site Scripting) Cheat Sheet." [Online]. Available: http://ha.ckers.org/xss.html

[143] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting Frame Busting : a Study of Clickjacking Vulnerabilities on Popular Sites A Survey of Frame busting," 2010. [Online]. Available: http://seclab.stanford.edu/websec/framebusting/framebust.pdf

[144] Samuraj Data AB, "Online viewer for PDF, PostScript and Word." [Online]. Available: http://view.samurajdata.se/

[145] Scansafe, "ANNUAL GLOBAL THREAT REPORT 2009," 2009. [Online]. Available: www.scansafe.com/downloads/gtr/2009_AGTR.pdf

[146] C. SEIFERT, R. STEENSON, I. WELCH, P. KOMISARCZUK, and B. ENDICOTTPOPOVSKY, "Capture A behavioral analysis tool for applications and documents," *Digital Investigation*, vol. 4, pp. 23–30, Sep. 2007.

[147] C. Seifert, "Know Your Enemy : Behind the Scenes of Malicious Web Servers," pp. 1–18, 2007. [Online]. Available: http://www.honeynet.org/papers/wek

[148] C. Seifert, V. Delwadia, P. Komisarczuk, and D. Stirling, "Measurement Study on Malicious Web Servers in the. nz Domain," in *Proc. ACISP '09 Proceedings of the 14th Australasian Conference on Information Security and Privacy*. Brisbane, Australia: Springer-Verlag, 2009, pp. 8–25.

[149] C. Seifert, I. Welch, and P. Komisarczuk, "Effectiveness of security by admonition: a case study of security warnings in a web browser setting," *(In)secure Magazine*, vol. 1, no. 9, 2006.

[150] ——, "Taxonomy of Honeypots - Technical Report CS-TR-06/12," Wellington, New Zealand, pp. 1251–1247, Nov. 2006. [Online]. Available: http://www.mcs.vuw.ac.nz/comp/Publications/archive/CS-TR-06/CS-TR-06-12.pdf

[151] ——, "Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots," in *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*. New York, New York, USA: ACM Press, 2008, p. 1426.

[152] ——, "Identification of Malicious Web Pages with Static Heuristics," in *2008 Australasian Telecommunication Networks and Applications Conference*. Adelaide, SA: Ieee, Dec. 2008, pp. 91–96.

[153] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing Android-Powered Mobile Devices Using SELinux," *IEEE Security & Privacy Magazine*, vol. 8, no. 3, pp. 36–44, May 2010.

[154] D. Shanley and McCann Erickson Advertising Ltd, "jgooglesafebrowsing," 2008. [Online]. Available: http://code.google.com/p/jgooglesafebrowsing/

[155] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang, "An empirical analysis of phishing blacklists," in *CEAS 2009: Sixth Conference on Email and Anti-Spam*, Mountain View, CA, 2009.

[156] R. Sheth, "Modern browsers for modern applications." [Online]. Available: http://googleenterprise.blogspot.com/2010/01/modern-browsers-for-modern-applications.html

[157] M. Silbey and P. Brundrett, "Understanding and Working in Protected Mode Internet Explorer," 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb250462(VS.85).aspx

[158] M. SiteAdvisor, "McAfee SiteAdvisor Software Website Safety Ratings and Secure Search." [Online]. Available: http://www.siteadvisor.com/

[159] S. Solomon, "BC woman killed by fake drugs bought online," *National Review of Medicine*, vol. 4, no. 13, 2007. [Online]. Available: http://www.nationalreviewofmedicine.com/issue/2007/07_30/4_policy_politics_13.html

[160] A. Sotirov and M. Dowd, "Bypassing browser memory protections," in *Black Hat USA*, 2008.

[161] Sourcefire, "ClamAV." [Online]. Available: http://www.sourcefire.com/security-technologies/ClamAV

[162] SpamHaus, "XBL." [Online]. Available: http://www.spamhaus.org/xbl/

[163] Squid-cache.org, "About Squid." [Online]. Available: http://www.squid-cache.org/Intro/

[164] Statcounter, "Top 12 Browser Versions on Oct 10 — StatCounter Global Stats," 2010. [Online]. Available: http://gs.statcounter.com/#browser_version-ww-monthly-201010-201010-bar

[165] Statowl.com, "Web Browser Plugin Market Share / Global Usage." [Online]. Available: http://www.statowl.com/plugin_overview.php

[166] Symantec, "Web Based Attacks," 2009. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/web_based_attacks_02-2009.pdf

[167] M. Takesue, *An HTTP Extension for Secure Transfer of Confidential Data*. Hunan: IEEE, Jul. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5197305

[168] The Apache Software Foundation, "ab - apache http server benchmarking tool - apache http server." [Online]. Available: http://httpd.apache.org/docs/2.0/programs/ab.html

[169] The Measurement Factory, "eCAP Home." [Online]. Available: http://www.e-cap.org/

[170] Toddml, "Spam and Malware Protection." [Online]. Available: http://blog.bit.ly/post/263859706/spam-and-malware-protection

[171] Trend Micro Inc, "Web Threat Spotlight - Stolen FTP Logins Open Door to Mass Compromise," pp. 1–2, 2009.

[172] ——, "Security Spotlight - PREDICTABLY UNPREDICTABLE FAKEAVS," pp. 1–5, 2010.

[173] Twitter.com, "Twitter." [Online]. Available: http://twitter.com/

[174] P. Uhley, "Setting up a socket policy file server — Adobe Developer Connection," 2008. [Online]. Available: http://www.adobe.com/devnet/flashplayer/articles/socket_policy_files.html

[175] Untangle, "product overview - multi-functional firewall software," 2010. [Online]. Available: http://www.untangle.com/Product-Overview

[176] A. Vance, "Advertising - On the Web, Ads Can Be a Security Hole - NYTimes.com," New York, NY, Usa, p. B5, 2009. [Online]. Available: http://www.nytimes.com/2009/09/15/technology/internet/15adco.html?_r=1

[177] VirusTotal, "Free Online Virus, Malware and URL Scanner." [Online]. Available: http://www.virustotal.com/stats.html

[178] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross site scripting prevention with dynamic data tainting and static analysis," in *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, vol. 42. San Diego, CA: Citeseer, 2007.

[179] W3counter, "w3counter - global web stats," 2010. [Online]. Available: http://www.w3counter.com/globalstats.php

[180] T. Wadlow and V. Gorelik, "Security in the browser," *Communications of the ACM*, vol. 52, no. 5, p. 40, May 2009.

[181] L. Wang and L. Qian, "An Alternative Means for Delivering Web Documents to Mobile Phones," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. Shanghai: IEEE, Sep. 2007, pp. 3093–3096.

[182] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, S. King, and Others, "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS06)*, San Diego, CA, 2006.

[183] D. Web, "Dr Web For Internet Gaterways Unix." [Online]. Available: http://products.drweb.com/gateway/unix/?lng=en

[184] WebKit, "Applications Using WebKit." [Online]. Available: http://trac.webkit.org/wiki/ApplicationsusingWebKit

[185] ——, "Companies and Organizations that have contributed to WebKit." [Online]. Available: http://trac.webkit.org/wiki/CompaniesandOrganizationsthathavecontributedtoWebKit

[186] ——, "The WebKit Open Source Project." [Online]. Available: http://webkit.org/

[187] C. Whittaker, B. Ryner, and M. Nazif, "Large-Scale Automatic Classification of Phishing Pages," in *Proc. 17th Annual Network and Distributed System Security Symposium, NDSS'10*, San Diego, CA, 2010. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/proceedings.shtml

[188] M. Winter, "Protecting the Intranet Against JavaScript Malware and Related Attacks," in *Detection of intrusions and malware, and vulnerability assessment: 4th international conference, DIMVA 2007*. Lucerne, Switzerland: Springer-Verlag New York Inc, 2007, p. 40.

[189] B. Witten, C. Landwehr, and M. Caloyannides, "Does open source improve system security?" *IEEE Software*, vol. 18, no. 5, pp. 57–61, 2001.

[190] C. Wuest, "Phishing In The Middle Of The Stream-Todays Threats To Online Banking," in *The Eighth Association of anti Virus Asia Researchers Conference, March*, no. September. Citeseer, 2006, pp. 1–17.

[191] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel, "SWAP: Mitigating XSS attacks using a reverse proxy," in *2009 ICSE Workshop on Software Engineering for Secure Systems*. Vancouver, BC: IEEE, May 2009, pp. 33–39.

[192] W. Yan and E. Wu, *Toward Automatic Discovery of Malware Signature for Anti-Virus Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 4, pp. 724–728.

[193] Y. Zhang, S. Egelman, L. Cranor, and J. Hong, "Phinding phish: Evaluating anti-phishing tools," in *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*. San Diego, CA: Citeseer, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.263&rep=rep1&type=pdf

[194] Zoho, "ZohoViewer - Online document viewer." [Online]. Available: http://viewer.zoho.com/home.do

[195] ZScaler, "State of the Web Q1 2010 A View of the Web From an End User s Perspective," pp. 1–19, 2010.

Appendix A

The HyperText Transfer Protocol

A.1 HyperText Transfer Protocol HTTP

A.1.1 General HTTP operation

The HyperText Transfer Protocol (HTTP) operates in a very simple request response fashion, assuming the client and server are already connected directly via TCP:

1. Client sends request chain.

2. Web server sends a response code with appropriate content, or error.

For example if a client wishes to request *index.html* from a server already connected to, the operation proceeds as in figure A.1 below.

Figure A.1: Direct connection in simplified HTTP transaction

This is a simplified version, as there are more fields and details that may be used depending on the specific version and implementation.

A.1.2 History and versions

The initial version of HTTP, version 0.9, allowed only the operation given above where a client could only request a resource and would always receive an HTML file in response. This was expanded upon in later versions and there are now two later versions still in use, versions 1.0[13]

and 1.1[50] (Although 1.1 is near universal, a few simple devices use 1.0 still where 1.1 additions are unnecessary). Version 1.0 was introduced in May 1996[13], and allowed the protocol to perform more complex operations than version 0.9.

Although some of these features were already in use unofficially, HTTP version 1.0 officially introduced the following features (among others):

- MIME media types - non-HTML content can be carried
- The POST request method - clients can send content to the server.
- Content encodings - compression or encryption of content
- Multipart responses
- Proxy server support
- Simple caching support - sending and handling expiry details.

There were some shortcomings in HTTP 1.0 however, and HTTP 1.1[50][51] improved upon 1.0. The main differences between version 1.0 and 1.1 are in the areas of[86]:

1. Extensibility - Additional headers and the ability to "UPGRADE" a request to a different set of protocols
2. Caching - Better cache control
3. Bandwidth Optimization - Partial requests and more detailed compression handling
4. Network Connection Management - TCP connection reuse and request pipelining. (Version 1.1 had full connection TCP connection set up and tear down for each request)
5. Message Transmission - Required overhaul to fit with network connection management (above).
6. Internet Address Conservation - Multiple hostnames can exist on one server as clients must specify the host
7. Error Notification - More comprehensive
8. Security, Integrity, and Authentication - Digest authentication, MD5 content integrity checks, cookies for state management.
9. Content Negotiation - Allowing the client to request specific content, such as by specific language

The additions in HTTP 1.1 made it a more complex protocol than version 0.9, but it is capable of accomplishing a more varied role. We shall now discuss its operation in more detail, discussing the operation only of HTTP versions 1.0 and 1.1. It is necessary to discuss both here, for while most modern browsers are 1.1 compatible, the squid Web proxy (see section 3.6.1) is only fully conformant to HTTP 1.0.

A.1.3 Request operation

Care must be taken here as the term *request* is a slight misnomer: it refers not simply to a client requesting a specific file, but to *any* message sent from an http client (such as a web browser) to an http server. This can include the sending of data via a POST method (discussed below). Furthermore, note that at this stage we are only discussing **direct** (i.e.*unproxied*) communications between client and server.

An HTTP request chain is the series of HTTP communicaions and headers sent from a client to a server. According to the definitions in the IETF Request For Comment (RFC)[51], an HTTP request chain contains the following separated by newlines (CRLF), with the body separated from the headers by a further blank line:

1. Request line
2. HTTP headers
3. Request body

The Request Line

The first line in an HTTP request, the request line, is of the form:

Method <**space**> **request-uri** <**space**> **http-version** <**CRLF**>

We will discuss each briefly in turn.

The HTTP Request Method parameter specifies which type of operation the client would like to perform with the web server. This will generally be either requesting a file, or placing some content to the web server, but there are many more options in the protocols. HTTP 1.0 allows three method types, while HTTP 1.1 offers an additional five types. Only three are directly relevant to this project:

- GET

- POST
- CONNECT

The GET request method is used to single request a file or resource from the web server. It is the most commonly used, and also conceptually the simplest method available in HTTP. It is available in all HTTP versions.

The POST request method requests are used to submit some content to be processed by a resource on the web serer. Normally this will be some input for a server side script. It is available from HTTP version 1.0 onwards and in this project this method is used for analysis of (and protection from) cross-site scripting or cross-site request forgery attacks (see sections 2.5.1 and 2.5.2).

The CONNECT request method is used to allow an intermediate web proxy (see sections 2.1.1 and 2.1.3) to dynamically switch to being a tunnel. It is available only in HTTP version 1.1. This method is relevant to the discussions here as it enables an intermediary proxy (that terminates and reestablishes a TCP connection) to tunnel encrypted content without needing to decrypt it. This is discussed in 2.1.3 below in more detail.

The request-uri parameter is given as a path relative to the root of the server when non-proxied. For example, to request (GET) the file http://www.example.com/pages/page1.html
The server will connect to the server at www.example.com and request:

GET /pages/page1.html ...

It will also send the required host-header of:

Host: www.example.com

The http-version parameter in the request line is used for backwards compatibility to let the server know which HTTP versions the client supports. Ordinarily this will be HTTP/1.1.

HTTP Headers

There are 19 header types in HTTP 1.1 [51], however, the only mandatory one in requests is the *HOST* header. The other headers are optional and generally irrelevant to this work, with three exceptions: proxy-auth, referrer, and user-agent.

The HTTP host header specifies the hostname to which the request is being sent. This allows a server to host multiple web pages sites on the same IP address and port. whereas previously it was only possible to run one web site per IP/Port combination.

The HTTP referrer header is used to specify the page on which the requested link was referenced; it should be empty if the address was typed directly. This feedback is useful for those running servers as it allows them to track things such as traffic origin, and some sites use it to prevent *deep linking*[1]. It also can be a potential privacy issue for users. User tracking can also be more pervasive with content hosted off the main server the user is accessing (such as advertisements), as this allows a larger proportion of the user's activities to be tracked.

The HTTP user-agent header contains the type and version of the software being used to make the request, normally a web browser. In modern browsers it also tends to contain information on the browser's plugins (discussed in section 2.5.5). This allows a server to serve up different content depending upon the browser being used, which is useful for page rendering between browsers which render a page differently. This is becoming a security risk to some users, as it allows precise Fingerprinting of the user's software for attacks such as drive by downloads (see section 2.5.4).

The HTTP proxy-authorization header is used to respond to a request from a web proxy for authorization (see section A.1.3); it was used in our testing to allow simple separation between "users" without requiring complicated transparent proxy rules and multiple systems.

Request Body

This is where any content that needs to be sent and cannot be sent in an HTTP request header is contained. This is only relevant this project with the POST method and is used for analysis of (and protection from) cross-site scripting or cross-site request forgery attacks (see sections 2.5.1 and 2.5.2).

[1] Where a single page or file on a web server is linked to on an external site - examples of this include placing an image off one site on another without permission. This not only uses bandwidth of the other party, but can have copyright implications.

A.1.4 Response operation

HTTP response operations are the communications sent from the HTTP server to an HTTP client, and are structured almost identically to HTTP requests, although the message contents differ. An *HTTP response chain* is the series of communications that is sent from the client to the server. According to the definitions in the RFCs [51], an HTTP response chain contains the following separated by newlines (CRLF) with the body separated from the headers by a further blank line:

1. Status line
2. HTTP response headers
3. Response (entity) headers and body

Status Line

The first line in an HTTP response, the status line is of the form:

http-version <**space**> **status-code** <**space**> **reason-phrase** <**CRLF**>

These are briefly discussed next.

HTTP-version is used to let the client know which HTTP version the server is using, mainly for backwards compatibility reasons. Ordinarily this will be HTTP/1.1, but with some limited functionality or single purpose web servers this could be HTTP/1.0.

The Status Code and Reason Phrase parameters are probably the most significant part of the response other than the body for clients. This is because the these handle not only the content, but also error handling. The status code is a 3 digit numeric number corresponding to a code from the protocol specification, where the first number denotes which class of status.

There are five classes of status code in the HTTP specification:

- 1xx Informational
- 2xx Success (200 - OK)
- 3xx Redirection e.g. 301 (moved permanently, 302 found)
- 4xx Client Error (407 - proxy authentication required)

- 5xx Server error

The full list of status and reason codes is in section 6.1.1 of RFC 2616[51], but the purposes of this work only a few statuses are relevant, namely 200, 301, 302, and 407.

The response status 200 - OK indicates success for the method it is in response to. In the case of a GET request then 200 response includes the requested resource in the response body (see section 2.7.4 below), and in the case of a POST request the response body describes or contains the result of the action.

The response statuses 301 - Moved Permanently, 302 - Found, and 303 - See Other are used to redirect clients to another location, and for the purposes of this project they are considered identical. If in response to a GET request then this response is normally handled transparently by web browsers which will then automatically request the resource from the new URL. If in response to a POST request then the redirection should not be automatic for 301 and 302 requests, but for a 303 redirection a client may automatically use the new resource location.

These statuses are important for this project as not only do we use them to redirect clients in our system, but many URL Shorteners (see section 2.4.3) also use them for transparent redirection.

The response status 407 - Proxy Authentication Required is returned by a proxy to a client before the proxy services the request if authentication is required to user the proxy. This response also gives an appropriate challenge for the client to use in any following authentication. The client is not required to respond, but should it do so it will need to use a proxy-authorization header (see section A.1.3 above).

Response Headers

In addition to the response status line, the server will send response headers that are used for many purposes, examples of which include assisting the client in rendering the content, or instructing caches along the way. For this project, the only response header directly used is the content-type header.

The HTTP content-type header indicates to a client what media type follows in the body of the response so that the client knows how to correctly handle it (if possible). For an HTML page this is *text/html*. Some other relevant media types for this project are given in table A.1

Content Media Type	Description
application/binary-stream	Any binary data not otherwise treated (base 64 encoded)
application/javascript	Javascript elements
application/msword	Microsoft Word Documents
application/pdf	PDF Documents
application/x-shockwave-flash	Shockwave Flash Elements
application/zip	Zip Archives
image/jpeg	JPEG Images
image/gif	GIF Images
image/png	PNG Images
text/plain	Any textual content not otherwise handled
text/html	HTML Content
text/css	Cascading Style Sheet (CSS) Content

Table A.1: Some relevant media types

Response Body

The response body contains content that is not carried in the headers of the response. For this project, the response body is used for giving the response to GET requests, for giving a redirection URL, and for giving the challenge for proxy-authorization requests. The majority of our project's security operations also require inspection and alteration of this section of an HTTP response.

Appendix B

Testbed Component Layout

Machine (OS)[a]	Component	(Interface) Address : Port
1 (U)	Firewall / Gateway (External Side)	(Eth0) 132.181.19.2
	Firewall / Gateway (Internal Side)	(Eth1) 10.1.1.1
1/2??	Clam Antivirus	10.1.1.1
2 (U)	Squid HTTP Proxy	10.1.1.2 3128
	SafeSquid HTTP Proxy	10.1.1.2 8080
3 (XP)	Greasyspoon ICAP Server	10.1.1.3 1344
	GreasySpoon Admin	N/A (client)
4 (U)	Mysql Server	10.1.1.5 3306
(U)	Free Radius Server	10.1.1.20 : 1812
5 (XP)	Capture HPC 1 (Unprotected)	10.1.1.10 : 902
(U)	CHPC Server (Unprotected)	10.1.1.10 : 7070
(U)	Web Polygraph Server	10.1.1.20
6 (XP)	Capture HPC 2 (Client Protection)	10.1.1.11 : 902
(U)	CHPC Server (Antivirus)	10.1.1.11 : 7070
(U)	Web Polygraph Server	10.1.1.20
7 (XP)	Capture HPC 3 (GreasySpoon ICAP)	10.1.1.12 : 902
(U)	CHPC Server (SafeSquid)	10.1.1.12 : 7070
(U)	Web Polygraph Client 1	10.1.1.20
8 (XP)	Capture HPC 4 (SafeSquid)	10.1.1.13 : 902
(U)	CHPC Server (Greasyspoon)	10.1.1.13 : 7070
(U)	Web Polygraph Client 2	10.1.1.22

[a] U = Ubuntu 10.04, XP = Windows XP professional Service Pack 3

[b] Bridged means ...

Table B.1: Testbed components and network details

Appendix C

Comcast System Requirements in full

4. Design Requirements

This section describes all of the requirements taken into consideration for the design of this system.

4.1. General

REQ1: TCP Port 80: The system should provide notifications via TCP port 80, the well-known port for HTTP traffic.

REQ2: Block Listing: It is possible that the HyperText Markup Language (HTML, [RFC1866]) or JavaScript [RFC4329] used for notifications may cause problems while accessing a particular website. Therefore, such a system should be capable of using a block list of website Uniform Resource Indicators (URIs, [RFC2396]) or Fully Qualified Domain Named (FQDNs, Section 5.1 of [RFC1035]) that conflict with the system, to instruct the system to not provide a notifications related to certain sites, in order to reduce any errors or unexpected results.

REQ3: Instant Messaging (IM): Some IM clients use TCP port 80 in their communications, often as an alternate port when standard, well-known ports do not work. This system should not conflict with or cause unexpected results for IM clients (or any other client types).

REQ4: Handling of Active Sessions: To the extent that a web notification system must temporarily route TCP port 80 traffic in order to provide a notification, previously established TCP port 80 sessions should not be disrupted and should be routed to the proxy layer.

REQ5: No TCP Resets: The use of TCP resets has been widely criticized, both in the Internet community generally as well as in [RFC3360]. As such, except for the case of unintentional errors, the use of TCP resets must be avoided.

REQ6: Non-Disruptive: The web notification system should not disrupt the end user experience, such as causing significant clients errors.

REQ7: Notification Acknowledgement: Once a user responds and acknowledges a notification, the notification should immediately stop.

REQ8: Non-Modification of Content: Such a system should not significantly alter the content of the HTTP response from any website the user is accessing.

REQ9: Unexpected Content: The system should transparently handle traffic for which it cannot provide a web notification.

Thus, widely varying content should be expected, and all such unexpected traffic should be able to be handled by the system without generating errors or unexpected results.

REQ10: No Caching: Web content must not be cached by the system.

REQ11: No Advertising Replacement or Insertion: The system must not be used to replace any advertising provided by a website, or insert advertising into websites where none was intended by the owner of a given website.

4.2. Web Proxy

REQ12: Open-Source Software: The system should use an open source web proxy server, such as Squid. (While it is possible to use any web proxy, the use of open source, and openly documented software is recommended.)

REQ13: ICAP Client: The web proxy server should have an integrated ICAP client.

REQ14: Access Control: Access to the proxy should be limited exclusively to the IP addresses of users for which notifications are intended, and only for limited periods of time. Furthermore, if a Session Management Broker (SMB) is utilized, as described in Section 5.1 below, then the proxy should restrict access only to the address of the SMB.

4.3. ICAP Server

REQ15: Request and Response Support: The system should support both request and response adaptation.

REQ16: Consistency: The system must be able to consistently provide a specific notification.

REQ17: Multiple Notification Types: The system must be able to provide many different types of notifications.

REQ18: Simultaneous Differing Notifications: The system must be able to simultaneously serve multiple notifications, including notifications of varying types, to different users. As a result, User A should be able to get the notification intended specifically for User A, at the same time that User B receives an entirely different notification, which was intended specifically for User B.

4.4. Messaging Service

REQ19: Messaging Service: The Messaging Service, as described in Section 5.1 below caches the notifications for each specific user. Thus, by caching the notification messages, the system may provide notifications without significantly affecting the web browsing experience of the user.

REQ20: Process Acknowledgements: The Messaging Service should process acknowledgements to properly remove entries from the cache and forward acknowledgements to the Messaging Service.

REQ21: Ensure Notification Targeting Accuracy: The Messaging Service must ensure that notifications are presented to the intended users.

REQ22: Keep Records for Customer Support: The Messaging Service should maintain some type of record that a notification has been presented and/or acknowledged, in case a user inquires with customer support personnel.

Appendix D

Threat Surveys and Studies

Source Type	Source	URL
General Vendor	Microsoft Security	http://www.microsoft.com/sir/
	IBM X-Force	http://xforce.iss.net//
Government agency and other non-vendor	OECD Report	http://www.oecd.org/dataoecd/53/34/40724457.pdf
	Australian House Standing Committe on Communications Inquiry into Cyber Crime	http://www.aph.gov.au/house/committee/coms/cybercrime/report.htm
	Internet Crime Complaint Center (IC3)	http://www.ic3.gov/media/annualreports.aspx
	US-Computer Emergency Response Team (US-CERT)	http://www.us-cert.gov/resources.html
	Anti Phishing Working Group	http://www.antiphishing.org/resources.html
Antivirus Vendor	Bit Defender	http://www.bitdefender.com/site/view/BitDefender-E-Threats-Landscape-Report.html
	ESET	http://www.eset.com/threat-center
	Fortiguard	http://www.fortiguard.com/ListOfArticles.html
	G Data	http://www.gdatasoftware.co.uk/about-g-data/press-centre/news.html
	Kaspersky	http://www.securelist.com/en/analysis
	McAfee	http://www.mcafee.com/apps/resource-library-search.aspx?region=us
	Sophos	http://www.sophos.com/security/
	Symantec	http://www.symantec.com/business/theme.jsp?themeid=threatreport
	Trend Micro	http://us.trendmicro.com/us/trendwatch/research-and-analysis/threat-reports/index.html
Other Security Vendors and Commercial Security Labs	AV Comparatives	http://www.av-comparatives.org/comparativesreviews
	Barracuda Labs	http://www.barracudalabs.com/research_resources.html
	Cascadia Labs	http://www.cascadialabs.com/clients.html#examples
	Dasient	http://blog.dasient.com
	M86 Security	http://www.m86security.com/labs/resources.asp
	Malware Intelligence	http://www.malwareint.com/docs.html
	Messagelabs	http://www.messagelabs.com/resources/mlireports
	Zscaler	http://www.zscaler.com/resourcesindustryreports.html

Appendix E

Configuration Settings

E.1 System Settings

E.1.1 Squid

squid.conf

```
#disable cache completely
cache deny all
icp_access deny all
htcp_access deny all

#Access control lists
acl manager proto cache_object
acl localhost src 127.0.0.1/32
acl to_localhost dst 127.0.0.0/8
acl localnet src 10.0.0.0/8
acl SSL_ports port 443

#Allow many ports, otherwise squid doesn't allow strange ports which we want for malware testing.
acl Safe_ports port 21
acl Safe_ports port 70
acl Safe_ports port 80
acl Safe_ports port 210
acl Safe_ports port 280
acl Safe_ports port 443
acl Safe_ports port 488
acl Safe_ports port 591
acl Safe_ports port 777
acl Safe_ports port 1025-65535
acl CONNECT method CONNECT

#Access Control Lists, so Squid knows who it is allowed to accept as clients.
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localnet
http_access allow localhost
http_access deny all

#The port squid operates on
```

```
http_port 3128

#Logging Directory
access_log /var/log/squid3/access.log squid

##ICAP Settings FOLLOW
#Enable ICAP
icap_enable on
#Stop Squid From bypassing ICAP after errors
icap_service_failure_limit -1
icap_service_revival_delay 30
#Not used, but doesn't hurt
icap_preview_enable on
#Send client IP and username to the ICAP server
icap_send_client_ip on
icap_send_client_username on
icap_client_username_header X-Authenticated-User
icap_client_username_encode on
#Icap Server address and details
icap_service greasyspoon_req reqmod_precache 0 icap://10.1.1.3:1344/greasyspoon_req
icap_service greasyspoon_resp respmod_precache 0 icap://10.1.1.3:1344/greasyspoon_resp
icap_class class_1 greasyspoon_req
icap_class class_2 greasyspoon_resp
icap_access class_1 allow all
icap_access class_2 allow all

#Miscellaneous settings follow
coredump_dir /var/spool/squid
```

E.1.2 Greasyspoon

greasyspoon.ini

```
############################################################
# ICAP OPTIONS parameters
# Only Values Changed form Defaultsa Shown
Options-TTL=180
Max-Connections=50
Preview=1000
Keep-Alive=enable
path=conf;lib;serverscripts;lib/mysql-connector-java-5.1.12-bin.jar;
maxtimeout = 60000
errorthreshold = 0
bypassonerror = true
```

icapserver.conf

```
proxyhost  10.1.1.2
proxyport  3128
tcp_lowlatency on
icap GreasySpoon * 1344 greasyspoon.ini
#Logging settings omitted
#Web editor settings omitted
#Admin Interface settings Omitted
```

Appendix F

Testing Configuration

F.1 Effectiveness Testing

F.1.1 CaptureHPC Server

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="config.xsd">
        <!-- version 2.6 -->
        <global collect-modified-files="true"
                client-default="iexplorebulk"
                client-default-visit-time="90"
                capture-network-packets-malicious="true"
                capture-network-packets-benign="false"
                send-exclusion-lists="false"
                terminate="true"
                group_size="5"
                vm_stalled_after_revert_timeout="180"
                revert_timeout="120"
                client_inactivity_timeout="120"
                vm_stalled_during_operation_timeout="600"
                same_vm_revert_delay="12"
                different_vm_revert_delay="24"
    />

        <exclusion-list monitor="file" file="FileMonitor.exl" />
        <exclusion-list monitor="process" file="ProcessMonitor.exl" />
        <exclusion-list monitor="registry" file="RegistryMonitor.exl" />

        <virtual-machine-server type="vmware-server" address="10.1.1.10" port
            ="902"
                username="user" password="password">
                <virtual-machine vm-path="C:\Virtual Machines\Win XP (
                    Unprotected)\Windows Xp Professional.vmx"
                                                client-path="C:\Progra~1\
                                                    capture\CaptureClient.
                                                    bat"
                                                username="user"
                                                password="password"/>
                <virtual-machine vm-path="C:\Virtual Machines\Win XP (
                    Unprotected 2)\Windows Xp Professional.vmx"
                                                client-path="C:\Progra~1\
                                                    capture\CaptureClient.
                                                    bat"
                                                username="user"
```

```
                                                    password="password"/>

-->     </virtual-machine-server>
</config>
```

F.1.2 Code used in URL Collection

F.1.3 Used twythonSearch.py to collect Urls to file

```
import twython.core as twython
import re
import random
import time
import googleTrending
import datetime

filename = "C:\Documents and Settings\user\Desktop\Capture Directories\urls.
    txt"

def is_ascii(s):
    return all(ord(c) < 128 for c in s)

def getSearchResultsList(searchTerm,resultSetMaxSize):
    """ Instantiate Twython with no Authentication """
    twitter = twython.setup()
    errorFound = False
    while(errorFound == False):
        try:
            raw_search_results = twitter.searchTwitter(searchTerm, rpp=
                resultSetMaxSize)
            errorFound = False
            searchResults = []
            for tweet in raw_search_results["results"]:
                if is_ascii(tweet["text"]):
                    searchResults.append(tweet["text"])
            return searchResults
        except:
            errorFound = True
            time.sleep(360)
            print "Sleeping to let the api calm down..."



def geturls(resultsSet):
    returnSet = []
    if resultsSet is not None:
        for tweet in resultsSet:
            urls = re.findall(r'(https?://\S+)', tweet)
            returnSet.extend(urls)
    return returnSet

def processTrends(trends):
    returnList = []
    trends = trends["trends"].values()[0]
    for trend in trends:
        if trend['name'].startswith('#'):
            if is_ascii(trend['name'][1:]):
                returnList.append(trend['name'][1:])
        else:
```

```
            if is_ascii(trend['name']):
                returnList.append(trend['name'])
    return returnList

def appendTrendUrlsToTrendsFile(filename, trendName, urls):
    f=open(filename, 'a')
    now = datetime.datetime.now()
    for url in urls:
        f.write(url + '\n')
    f.close()

def appendCommentLineToTrendUrlsToTrendsFile(filename, commentLine):
    f=open(filename, 'a')
    now = datetime.datetime.now()
    f.write("#Written: " + str(now) + "\n")
    f.write(commentLine + '\n')
    f.close()

def main():
    twitter = twython.setup()
    totalUrls = 0
    while True:
        try:
            twitterCurrentTrendsRaw = twitter.getCurrentTrends()
            twitterDailyTrendsRaw = twitter.getDailyTrends()
            twitterWeeklyTrendsRaw = twitter.getCurrentTrends()

            twitterCurrentTrends = processTrends(twitterCurrentTrendsRaw)
            twitterDailyTrends = processTrends(twitterDailyTrendsRaw)
            twitterWeeklyTrends = processTrends(twitterWeeklyTrendsRaw)

            googleTrends = googleTrending.getTrends()

            print "Found " + str(len(googleTrends)) + " Google Trends."
            print "Found " + str(len(twitterCurrentTrends)) + " Current
                Twitter  Trends."
            print "Found " + str(len(twitterDailyTrends)) + " Daily Twitter
                Trends."
            print "Found " + str(len(twitterWeeklyTrends)) + " Weekly Twitter
                  Trends."

            trendsList = googleTrends + twitterCurrentTrends +
                twitterDailyTrends + twitterWeeklyTrends
            print "Found " + str(len(trendsList)) + " Total Trends."

            commentLine = "#All Trends: "
            for trend in trendsList:
                commentLine = commentLine + ", \"" + trend + "\""
            appendCommentLineToTrendUrlsToTrendsFile(filename, commentLine)

            for trend in trendsList:
                searchResults = getSearchResultsList(trend + str(" http"),
                    60)
                tempUrls = geturls(searchResults)
                numFound = len(tempUrls)
                commentLine = "#Trend: \"" + trend + "\", " + str(numFound) +
                     " results."
                appendCommentLineToTrendUrlsToTrendsFile(filename,
                    commentLine)
                appendTrendUrlsToTrendsFile(filename, trend, tempUrls)
                totalUrls = totalUrls + numFound
```

```
                print "Found " + str(len(tempUrls)) + " urls for " + trend +
                    ", for a total of " + str(totalUrls) + " so far.."
        except:
            print "Exception ocurred... Waiting"

        print "Waiting an hour or so before updating...",
        randNum = random.randint(3000,4000)
        print " Waiting " + str(randNum) + "seconds..."
        delay = randNum
        time.sleep(delay)

if __name__ == "__main__":
    main()
```

F.1.4 To collect Google Trends googletrending.py

```
import urllib
import re
from BeautifulSoup import BeautifulSoup

def getTrends():
    trends = []
    feedURL = "http://www.google.com/trends/hottrends/atom/hourly"

    fp = urllib.urlopen(feedURL)
    content = fp.read()

    reg = r"(<ol).*?(ol>)"
    regx = re.compile(reg,re.DOTALL)
    results = regx.search(content)
    content = results.group(0)

    soup = BeautifulSoup(content)
    tagLinks = soup.findAll("a")
    for tag in tagLinks:
        trends.extend(tag.contents)

    return trends
```

F.2 Performance Testing Automation

The following was used, a near identical version (proxy parameter removed from the apachebench arguments) was used in on non-proxied tests .

```
#!/bin/bash
datetime=`date +%F-%H%M`
serveraddress="http://10.1.1.11/"
durations=(60)
numreqs=(10000 100000)
concurrencies=(1 10 100 200 400)
fileexts=(".html" ".file")
filesizes=("1KB" "10KB" "100KB" "1MB" "10MB")
proxtype="proxied"
proxip='10.1.1.2:3128'

if [ $# -ne 1 ]; then
        echo "You must enter a test designator"
        echo "usage: <scriptname> <testdesignator>"
        exit
```

```
fi

parentfoldername="proxied_${1}"
echo "making␣folder␣$parentfoldername␣"
mkdir $parentfoldername
cd $parentfoldername
mkdir results
cd results
echo "Making␣folder␣$datetime}"
mkdir $datetime
cd $datetime

for duration in ${durations[*]}
do
        for reqcount in ${numreqs[*]}
        do
                for concurrency in ${concurrencies[*]}
                do
                        for filext in ${fileexts[*]}
                        do
                                for filesize in ${filesizes[*]}
                                do

echo "Testing␣$filesize␣$filext␣file␣for␣$duration␣seconds␣with␣$reqcount␣requests␣and␣
    concurrency␣of␣$concurrency."

#echo "running: ab -r -n $reqcount -c $concurrency -t $duration -X $proxip -gdata.${
    filesize}${filext}.${duration}sec.${reqcount}reqs.conc${concurrency}.${proxtype}.
    txt   -e summary.${filesize}${filext}.${duration}sec.${reqcount}reqs.conc${
    concurrency}.${proxtype}.csv    ${serveraddress}$filesizes}${filext} >
    ab_output_${filesize}${filext}.${duration}sec.${reqcount}reqs.conc${concurrency}.$
    {proxtype}.log"

ab -r -n $reqcount -c $concurrency -t $duration -X $proxip \
-gdata.${filesize}${filext}.${duration}sec.${reqcount}reqs.conc${concurrency}.${
    proxtype}.txt \
-e summary.${filesize}${filext}.${duration}sec.${reqcount}reqs.conc${concurrency}.${
    proxtype}.csv  \
${serveraddress}${filesize}${filext} \
> ab_output_${filesize}${filext}.${duration}sec.${reqcount}reqs.conc${concurrency}.${
    proxtype}.log
                                done
                        done
                done
        done
done
cd ..
cd ..
cd ..
echo "Results␣saved␣in␣${parentfoldername}/results/${datetime}."
```

Appendix G

Database Schemas

G.1 usermessages

```
CREATE TABLE userMessages(
  id int(11) unsigned NOT NULL auto_increment,
  UserName varchar(64) NOT NULL,
  MessageContents varchar(253) NOT NULL default '',
  PRIMARY KEY (id),
  KEY UserName (UserName(32))
) ;

CREATE TABLE globalUserMessages(
  id int(11) unsigned NOT NULL auto_increment,
  MessageContents varchar(253) NOT NULL default '',
  PRIMARY KEY (id),
) ;

CREATE TABLE userContentSettings(
  id int(11) unsigned NOT NULL auto_increment,
  UserName varchar(64) NOT NULL,
  SettingName varchar(253) NOT NULL default '',
  SettingValue varchar(64) NOT NULL default   ,
  PRIMARY KEY (id),
  KEY UserName (UserName(32))
) ;

CREATE TABLE userOverlaySettings(
  id int(11) unsigned NOT NULL auto_increment,
  UserName varchar(64) NOT NULL,
  SettingName varchar(253) NOT NULL default '',
  SettingValue varchar(64) NOT NULL default   ,
  PRIMARY KEY (id),
  KEY UserName (UserName(32))
) ;
```

Appendix H

Example Greasyspoon Code

H.1 Helper Code

H.1.1 The HTML Inserted to place a Message

```
<!-- Javascript functions to hide and show the overlay -->
<script type="text/javascript">
  function hideOverlay(){
    document.getElementById('ICAPoverlay').style.display = 'none';
    document.getElementById('ICAPoverlayTab').style.display = 'block';
  }
  function showOverlay(){
    document.getElementById('ICAPoverlay').style.display = 'block';
    document.getElementById('ICAPoverlayTab').style.display = 'none';
  }
</script>

<!--The Main container DIV that covers the whole page -->
<div id="ICAPoverlay" style="position:fixed; bottom:0; left:0; width:100%;
    height:40px; background-color:#FFDDCB; z-index:99999">
  <!-- Start Message -->
  <div style="width:80\%; position:absolute; left:0px; font-weight:bold; font
      -size:large;">

    Welcome username <a href="#">Link to click.</a><br>
  </div>

  <div style="width:20%; position:absolute; right:10%;">
    <button type="button">And a button to press!</button>
  </div>

  <!-- End Message and add Contro Overlay -->

  <div style="position:fixed; bottom:0; right:0; width:10%; height:40px;
      background-color:#FFDDCB; z-index:99999">
    <a href="#" onClick="javascript:hideOverlay();">Minimize Warning</a>
  </div>
</div>

<!-- The DIV that shows as a little tab down the bottom -->
<div id="ICAPoverlayTab" style="display:none; position:fixed; bottom:0; right
    :0; width:10%; height:40px; background-color:#FFDDCB; z-index:99999">
  <a href="#" onClick="javascript:showOverlay();">Show Warning Overlay</a>
</div>
```

H.1.2 DBMethods.java

```
package gspis;
import java.sql.*;

public class DBMethods{
  public static String getMessages(String dbClassName,String host,String
      database,String user,String password,String SQLSelect, String fieldName
      ) throws SQLException,Exception{
     String globalMessages = "";
     try{
       //load the jdbc classes and create class factory
       Class.forName(dbClassName);
       String dburl = "jdbc:mysql://" + host + ":3306/" + database;
       Connection dbconn = DriverManager.getConnection(dburl,user,password);
       Statement stmt = null;
       ResultSet rs = null;

       stmt = dbconn.createStatement();
       //Potentially Vulnerable to SQL Injection if care is not taken in the
       //select statement used, must sanitise in the script that calls this.
       //Ideally should move to prepared statements or similar.
       rs = stmt.executeQuery(SQLSelect);
       rs.last();
       if(rs.getRow() > 0 ){
         rs.first();
         globalMessages = rs.getString(fieldName);
         }
       dbconn.close();
       return globalMessages;
    }catch(SQLException ex){
      throw ex;
    }
    catch (Exception e){
      throw e;
    }
  }
}
```

H.2 Sample GreasySpoon Scripts

H.2.1 Clam Antivirus Scan

```
#rights=ADMIN
//-----------------------------------------------------------------------
// ==ServerScript==
// @name            antivirus
// @status on
// @description
// @include         .*
// @timeout  5000
// ==/ServerScript==
// ----------------------------------------------------------------------
// Note: use httpMessage object methods to manipulate HTTP Message
// use debug(String s) method to trace items in service log (with log level
    >=FINE)
// ---------------
import java.io.*;
```

```
import java.net.*;
//Set this to true to display a lot of debugging messages on the console...
private static boolean VERBOSEMODE = false;
private static String CLAMAV_IP = "10.1.1.2";
private static int CLAMAV_PORT = 3310;

 public void main(HttpMessage httpMessage){

     //If a null response then exit
     if (httpMessage.getUnderlyingBytes() == null){
         if(VERBOSEMODE) System.out.println("\n\n\nUnderlying bytes null");
         return;}
     else if(httpMessage.getUnderlyingBytes().length == 0){
         if(VERBOSEMODE) System.out.println("Underlying bytes empty");
         return;
      }
    //Our Initial Control Networking Variables
     Socket controlSocket = null;
     PrintWriter controlOut = null;
     BufferedReader controlIn = null;
    //Our Initial ByteData Networking Variables
     Socket dataSocket = null;
     OutputStream dataOut = null;

     try{
         if(VERBOSEMODE) System.out.println("Making Control Socket Connection
             ");
         String body ="";//The body variable to use if we rewrite the body
             later on

         //Open our clam AV control connection
         controlSocket = new Socket(CLAMAV_IP,CLAMAV_PORT);
         controlOut = new PrintWriter(controlSocket.getOutputStream(),true);
         controlIn = new BufferedReader(new InputStreamReader(controlSocket.
             getInputStream()));

        //Tell it we wish to STREAM a file
         controlOut.println("STREAM");

         //Read the port we are to use
         String port = controlIn.readLine();
         port = port.substring(port.lastIndexOf(' ') + 1 );
         int dataPort = Integer.parseInt(port);
         if(VERBOSEMODE) System.out.println("Av scan port: " + dataPort);

         if(VERBOSEMODE) System.out.println("Opening Data Port: ");
         //Open the Data socket to STREAM to
         dataSocket = new Socket(CLAMAV_IP,dataPort);
         dataOut = dataSocket.getOutputStream();

         if(VERBOSEMODE) System.out.println("Sending Content for scan...");
         //Get our raw content and send it
         byte[] bodyBytes = httpMessage.getUnderlyingBytes();
       if(VERBOSEMODE) System.out.println("Content Length:" + httpMessage.
           getUnderlyingBytes().length);
         dataOut.write(bodyBytes);
       dataSocket.close();

         if(VERBOSEMODE) System.out.println("Readng response...");
         //Read the response
         String response =  controlIn.readLine();
```

```
        if(VERBOSEMODE) System.out.println("Clamd says: " + response);

        //if we find something
        if(response.indexOf("FOUND") != -1){
         //Rewrite the message to let the user know it is blocked
          httpMessage.deleteHeader("Content-Type");
          httpMessage.addHeader("Content-Type","text/html; charset=UTF-8");
          body = "A Virus was found, File blocked!\n<br>";
          body += "Clam Antivirus reports: " + response + "<br />";
          httpMessage.setBody(body);

          System.out.println("Clamd found something: " + response);
        }else{
          if(VERBOSEMODE) System.out.println("Nothing found: " + response);
        }

        //Clean up
        dataOut.close();
        controlSocket.close();

    }catch(UnknownHostException e){
        //No host!
        System.out.println("Host error!" + e);
    }catch(IOException e){
        //No IO!
        System.out.println("IO Error!" + e);
    }

}
```

H.2.2 Block PDF by file header signature

```
#rights=ADMIN
//-------------------------------------------------------------------------
// ==ServerScript==
// @name            blockpdfbysig
// @status off
// @description
// @include         .*
// @exclude
// @responsecode    200
// ==/ServerScript==
// -------------------------------------------------------------------------
// Note: use httpMessage object methods to manipulate HTTP Message
// use debug(String s) method to trace items in service log (with log level
    >=FINE)
// ----------------

// ----------------
import java.util.concurrent.ConcurrentHashMap;
import java.util.Arrays;
public void main(HttpMessage httpMessage){
    try{

        //System.err.println(httpMessage.getResponseHeaders());
        //System.err.println(httpMessage.getBody());
        String body = httpMessage.getBody();
        ConcurrentHashMap<String, Object> sharedCache = httpMessage.
            getSharedCache();
        byte[] contentBytes = httpMessage.getUnderlyingBytes();
        byte[] signature = {'%','P','D','F','-'};
        byte[] newContentBytes = new byte[signature.length];
        System.arraycopy(contentBytes,0,newContentBytes,0,signature.length);
        boolean matches = true;
        for (int i =0;i<signature.length;i++){
             if (signature[i] != newContentBytes[i]){
                 matches = false;
                 break;
                 }
        }
        if (matches || (httpMessage.getResponseHeader("Content-Type") == "
            application/pdf")){
            httpMessage.deleteHeader("Content-Type");
            httpMessage.addHeader("Content-Type","text/html; charset=UTF-8");
            body = "Sorry, PDF Files are blocked!";
            httpMessage.setBody(body);
            System.out.println("match on sig");

       }else{
            httpMessage.deleteHeader("Content-Type");
            httpMessage.addHeader("Content-Type","text/html; charset=UTF-8");

            //body = "NOWAI~: ";
            //body += "header was:" + new String(newContentBytes);
            //body += "\nand signature was: " + new String(signature);
            //httpMessage.setBody(body);
            System.out.println("no match on sig, sig was: " + new String(
                signature) + " header was:" + new String(newContentBytes));
        }
        // httpMessage.toJson();
```

```
      httpMessage.minify();
    }catch (Exception e){
       debug(e.getMessage());
       httpMessage.setBody(e.getMessage());
    }
}
```

H.2.3 Redirect PDF to google's online viewer

```
#rights=ADMIN
//-----------------------------------------------------------------------
// ==ServerScript==
// @name            onlinedocumentviewer
// @status on
// @description
// @include         .*pdf
// @exclude
// @timeout    300
// ==/ServerScript==
// -----------------------------------------------------------------------
// Note: use httpMessage object methods to manipulate HTTP Message
// use debug(String s) method to trace items in service log (with log level
    >=FINE)
// ----------------

// ----------------
public void main(HttpMessage httpMessage){
    String[] extensions = {"pdf","ppt","doc","docx"};
    boolean VERBOSEMODE = false;
    String url = httpMessage.getUrl();
    boolean redirect = false;
    String referrer = httpMessage.getRequestHeader("Referer");

    if(VERBOSEMODE) System.out.println("Endswith pdf?: " + url.toLowerCase().
        endsWith("pdf"));
    if(!url.contains("docs.google.com") && !referrer.contains("docs.google.
        com") ){
        for(int i = 0;i < extensions.length;i++){
            if(VERBOSEMODE) System.out.println("Checking " + url + " for: " +
                extensions [i]);
            if(url.toLowerCase().endsWith(extensions[i]))
                {
                redirect = true;
                if(VERBOSEMODE) System.out.println("Url needs to be
                    redirected... ");
                break;
                }
        }
        if (redirect == true){
             if(VERBOSEMODE) System.out.println("Redirecting Url to docs
                 viewer: " + url);
             String encodedPageUrl = java.net.URLEncoder.encode(url);
             String redirectUrl = "http://docs.google.com/viewer?url=" +
                 encodedPageUrl ;
             String redirectHeader = "HTTP/1.0 302 Found" + "\r\n" + "
                 LOCATION:" + redirectUrl + "\r\n";// + "PRAGMA: no-cache\r\n
                 ";
             httpMessage.setHeaders(redirectHeader);
             //httpMessage.setBody(redirectHeader);
```

```
                //System.out.println(httpMessage.getBody());
            }else{
                    if(VERBOSEMODE) System.out.println("Url ignored for
                        extensions): " + url );
                }
        }else{
            if(VERBOSEMODE) System.out.println("Url google docs already: " + url
                );
        }
}
```

Appendix I

Sample Results

I.1 Sample Drive by download steps taken from an Capture-HPC Log in the experiment

Some of the steps undertaken by a drive by download that used a weak version of Java to get onto the system and start itself without user interaction are given below. Only the various executions of different binaries are shown here, but during the steps shown settings and files were changed to make things automatically start, temporary files and cookies were deleted (so users would need to re-login to websites) and similar.

```
Internet Explorer -> Java Web Start
Java Web Start -> Java
Java -> C:\\WINDOWS\\Temp\\0.14564161834687983.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\teste1_p.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\q1.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\miraage.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\fFollower.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\avto.exe
teste1_p.exe -> C:\\WINDOWS\\lsass.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\6_ldry3no.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\4_pinnew.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\2_load.exe
6_ldry3no.exe -> C:\\WINDOWS\\system32\\sdra64.exe
avto.exe -> C:\\WINDOWS\\svc.exe
0.14564161834687983.exe -> C:\\WINDOWS\\Temp\\1your_exe.exe
miraage.exe -> C:\\WINDOWS\\system32\\jkkhef.dll
0.14564161834687983.exe -> opeA.exe
0.14564161834687983.exe -> 1276966669.exe
0.14564161834687983.exe -> 1_goo.exe
1your_exe.exe -> ijwxv.exe
1your_exe.exe -> wljrkxt.exe
```

I.2 ApacheBench Data

Test parameters: 10,000 Requests, Concurrency 1

	1KB			10KB			100KB			1MB			10MB		
Script Class	Reqs / s	Bandwidth (KB/s)	Mean Req Time (ms)	98% time (ms)	Reqs / s	Bandwidth (KB/s)	Mean Req Time (ms)	98% time (ms)	Reqs / s	Bandwidth (KB/s)	Mean Req Time (ms)	98% time (ms)	Reqs / s	Bandwidth (KB/s)	Mean Req Time (ms)	98% time (ms)
No Proxy, No ICAP	2790	3548	0	0	1850	19009	1	1	597	59819	2	2	72	74013	14	14	11	111739	92	101
Proxy, No ICAP	1253	1711	1	1	1031	10694	1	1	348	34952	3	3	44	45177	23	27	8	82129	125	156
Proxy, ICAP, No Scripts	504	688	2	2	460	4764	2	2	267	26788	4	4	56	57124	18	23	7	69985	146	177
Proxy, ICAP Request Header Insertion	407	556	2	3	384	3978	3	3	243	24400	4	5	54	55553	18	24	7	69069	148	163
Proxy, ICAP Request Header Inspection / Alteration	473	647	2	2	443	4595	2	2	261	26207	4	4	55	56611	18	23	7	68193	150	187
Proxy, ICAP Request URL Conditional Change	471	643	2	2	436	4525	2	3	258	25925	4	4	55	56619	18	23	7	68196	150	182
Proxy, ICAP Request to Response Modification	410	560	2	3	381	3945	3	3	237	23772	4	5	54	54796	19	24	7	67070	153	187
Proxy, Icap Response Content Insertion (static)	35832	505	3	3	291	3029	3	4	118	11819	8	10	17	17843	57	65	2	19342	484	702
Proxy, Icap Response Content Insertion (Database - Non-caching)	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err	Err
Proxy, Icap Response Content Insertion (Database - Caching)	Err	Err	Err	Err	122	1419	8	9	44	4442	23	24	6	5779	177	207	1	5688	1507	6673
Proxy, ICAP Antivirus Scanning	246	335.99	4	5	202	2090	5	6	93	9354	11	14	15	15353	67	82	2	16331	552	1069
Proxy, ICAP Multiple Operations	187	263	5	6	158	1644	6	7	75	7579	13	16	12	12450	82	119	2	14324	652	1161
4-12 Mean	5432	501	3	3	302	3153	4	5	166	16687	9	10	34	34375	57	71	4	41027	475	1291